A Framework for Distributed Web-based Microsystem Design

by

Debashis Saha

B.Tech(Hons), Indian Institute of Technology, Kharagpur (1996)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1998

Author .....
Department of Electrical Engineering and Computer Science
February 1, 1998

Certified by .....
Anantha P. Chandrakasan
Assistant Professor of EECS
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chair, Department Committee on Graduate Students

A Framework for Distributed Web-based Microsystem Design

by

Debashis Saha

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

The increasing complexity of microsystem design mandates a distributed and collaborative design environment. The high integration levels call for tools and generators that allow exploration of the design space irrespective of the geographical or physical availability of the design tools.

The World Wide Web serves as a desirable platform for distributed access to libraries, models and design tools. The rapid growth and acceptance of the World Wide Web has happened over the same time period in which distributed object systems have stabilized and matured. The Web can become an important platform for VLSI CAD, when the distributed object technologies (e.g, CORBA) are combined with the Web technologies (e.g., HTTP, CGI) and Web-aware object oriented languages (e.g., Java).

In this thesis, a framework using the Object-Web technologies is presented, which enables distributed Web based CAD. The Object-Web architecture provides an open, interoperable and scalable distributed computing environment for microsystem design, in which Web based design tools can efficiently utilize the capabilities of existing design tools on the Web to build hierarchical Web tools. The framework includes the infrastructure to store and manipulate design objects, protocols for tool communication and WebTop, a Java hierarchical schematic/block editor with interfaces to distributed Web tools and cell libraries.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Assistant Professor of EECS

Acknowledgments

- I would like to express my deepest gratitude to my advisor, Anantha Chandrakasan, for his support, guidance and suggestions throughout the project. Anantha has always inspired and motivated me. It has been a wonderful and gratifying experience to have him as my advisor.
- My sincere thanks to my office mates, Abram Dancy, Duke Xanthapoulos, Gangadhar Konduri, Jim Goodman, Raj Amirtharajah, SeongHwan Cho, Vadim Gutnik and Wendi Rabiner for providing a wonderful environment to grow and develop myself.
- My appreciation and thanks to David Liebson for his assistance in the implementation of WebTop.
- My special thanks to Prof. Don Troxel, Prof. Duane Boning, Michael McIlrath and members of the DDF group for their feedback and suggestions.
- I gratefully acknowledge Defense Advanced Research Projects Agency for supporting my graduate study (DARPA Contract DABT63-95-C-0088).
- And most importantly, I would like to thank my parents and my brother, Sona, for their constant support, love and encouragement. Their blessings and sacrifice have made me what I am today.

Contents

1 Introduction **13**

1.1 CAD Challenges for Designing "Systems-on-a-chip" 13

1.2 The Web and the Internet 15

1.3 Distributed Microsystem Design Framework 17

1.4 The Model of Web based CAD 18

1.5 Meta Web tools 20

1.6 Related Work 22

2 Web based Technologies **27**

2.1 The Hypertext Transport Protocol 27

2.1.1 Connection 28

2.1.2 Request 28

2.1.3 Response 30

2.1.4 Disconnection 31

2.2 The Common Gateway Interface (CGI) 31

2.3 Java 34

2.4 Distributed Object Technologies 36

2.4.1 CORBA 36

2.4.2 Java RMI 41

2.5 Summary 43

3 Framework for Distributed Web-based CAD **45**

3.1 Object-Web Architecture 45

3.2 Infrastructure for Hierarchical Web-based CAD 48

3.3 HTML Forms and multipart/form-data 51

3.4 The Application Tier - Web based Point Tools 54
3.5 CGI back-end Tools 55
3.6 Hierarchical Web Tools 57
3.7 Java and Web Tools 58
3.8 Example of Web based Hierarchical Tools 60

4 WebTop: Distributed Microsystem Design Center 65
4.1 Overview of the Distributed Design Center 66
4.2 WebTop: Implementation and New Features 70
4.2.1 Vectored Pins and Bus 70
4.2.2 Vectored Instantiation 71
4.2.3 Strength of Net Types in Verilog 73
4.2.4 Behavioral Views as URL's and local files 73
4.2.5 CellServer to Save Files Remotely 74
4.2.6 Dynamic Cell Properties 75
4.2.7 Other Modifications 75
4.3 Distributed Tool Integration 77
4.3.1 Integration with Pythia 77
4.3.2 Integration with WebSpice 77
4.3.3 Integration with PowerPlay 78
4.3.4 Integration of a Buffer Generator 80

5 Design Examples 85
5.1 24 bit Ripple Carry Adder 85
5.2 Design of the IDCT 88

6 Summary and Future Work 93
6.1 Future Work 95

A Verilog Listing 97
A.1 Verilog Listing of Cells in 24 bit Ripple Carry Adder Example 97
A.1.1 The Sum Cell: fa_sum 97
A.1.2 The Carry Cell: fa_carry 98
A.1.3 The Inverter Cell: inv 99

A.1.4 The Top Level Cell: test 100

B Vectored Pin and Bus Connections **103**

List of Figures

1-1 Low Power Single Chip Multimedia Processor 14
1-2 Distributed CAD Environment 16
1-3 Distributed Microsystem Design Framework 17
1-4 Partitioning Web Tools as Clients and Servers 18
1-5 Hierarchical Web based design tools 21

2-1 The components of a simple WWW interaction 28
2-2 Data Flow during execution of a CGI Script 32
2-3 JAVA: More than a language 35
2-4 The OMA Reference Architecture 38
2-5 The CORBA Communication mechanism 39
2-6 The Structure of the Object Request Broker Interface [18] 40
2-7 The RMI Mechanism 42

3-1 Object-Web Architecture for Distributed Computing 46
3-2 Slicing of Applications based on Inputs and Outputs 50
3-3 Data flow in Hierarchical tools 57
3-4 The Proxy Mechanism 60
3-5 Snap shot of Pythia 62

4-1 WebTop: Library Manager 67
4-2 WebTop: Distributed Cell access 68
4-3 WebTop: Netlister 69
4-4 WebTop: Distributed Tool Integration 70
4-5 Netlisting Vectored Instances 73
4-6 The save in URL Dialog 75

4-7 Property Box 76
4-8 WebTop - PowerPlay Integration 79
4-9 Source of RemoteGen.java 80
4-10 Source of RemoteGenServer.java 81
4-11 Source of RemoteGenImpl.java 82
4-12 Client invocation on Remote Object 83
4-13 Schematics of a Buffer Generated by Remote Object 83

5-1 Schematic of the Sum Cell 86
5-2 Schematic of the Adder Cell 87
5-3 Schematic of the 24 bit Adder Cell 88
5-4 Schematic of the Top Level Test cell 89
5-5 Schematic of the Top Level IDCT Chip 91
5-6 Schematic of serializer_s1_new 91
5-7 Schematic of serializer_s1_2reg 92
5-8 Schematic of mux2_pass 92

B-1 CaseI: Schematics 104
B-2 CaseI: Netlist 104
B-3 CaseI: Schematics 105
B-4 CaseII: Netlist 105
B-5 CaseIII: Schematics 106
B-6 CaseIII: Netlist 106
B-7 CaseIV: Schematics 107
B-8 CaseIV: Netlist 107

Chapter 1

Introduction

"Managing complexity is of extreme importance. Design of a modern microprocessor, ..., typically takes place at several locations separated quite widely geographically. We can no longer get all the engineers to work on a project at one place, connected with an extensive computer network and a lot of tools", - Gordon Moore

The design of future high-performance VLSI systems will require a distributed design and verification methodology due to the diverse expertise required at various levels of abstraction. The emergence of *"systems-on-a-chip"* with more than tens of millions of transistors on a single chip calls for a distributed and collaborative design framework that will facilitate easy and fast information flow. The high integration levels will require tools and generators that allow exploration of the design space irrespective of the geographical or physical availability of the design tools.

1.1 CAD Challenges for Designing "Systems-on-a-chip"

Deep sub-micron scaling enables the integration of 10's of millions of transistors on a single chip. This allows the implementation of an entire system on a chip. Designing systems-on-a-chip increases both the complexity of design and the issues of management of the design process manifolds. The increased complexity in the microsystem design process poses new challenges to the VLSI CAD community. New CAD tools must be available which will be capable of handling the increased complexity. Keeping aside the challenging issues in the circuit and system design, it is an arduous task to be able to integrate such huge designs and have tools which efficiently test and verify the

designs. Therefore, it is desirable that we break up the problem in a modular way and have a collaborative and distributed design methodology.

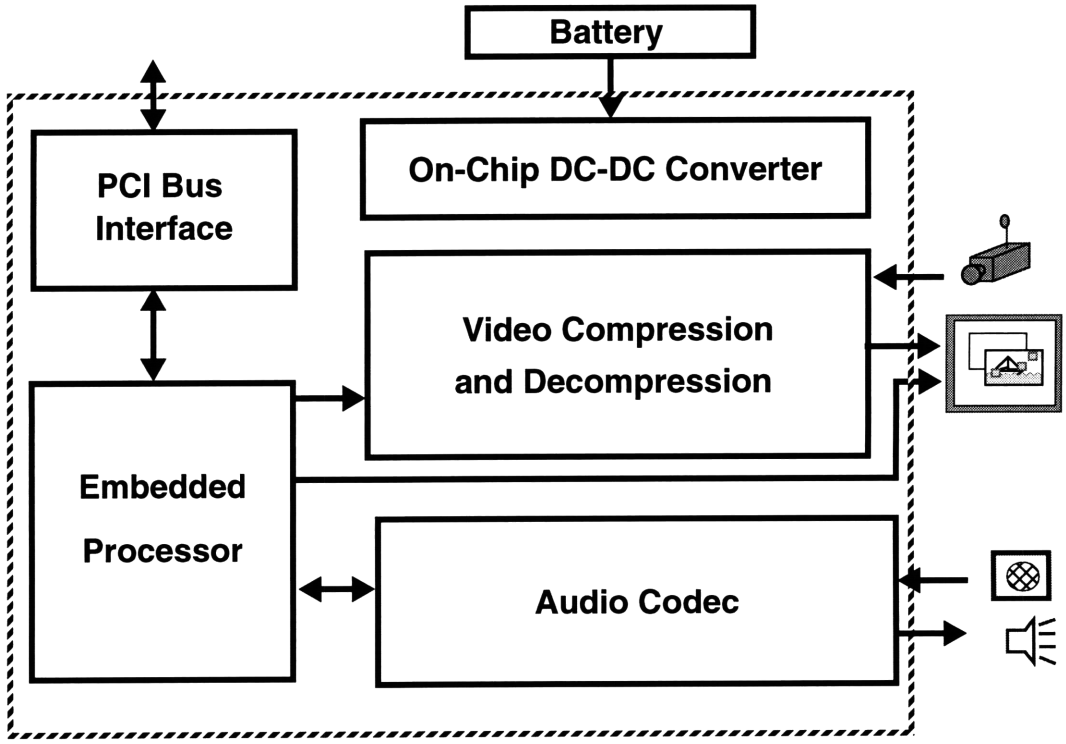

Figure 1-1: Low Power Single Chip Multimedia Processor

To explore some of the issues in the design of a CAD environment for future systems-on-a-chip, a multi-university project, Vela [24], is underway. Figure 1-1 shows a block diagram of the design driver, which is a low power single chip multi-media processor. The chip consists of various modules which require diverse expertise. For example, the chip has a DC-DC converter, an embedded processor, a video decompression module and many other modules. The design of each of these individual module requires diverse knowledge base and expertise. The design should exploit existing modules and libraries that are available at different places. It is therefore advantageous to have a distributed design methodology, which helps the design process by utilizing the existing technical know-how. This not only reduces the complexity of the design by providing a more modular approach, but also helps shorten the *design-turn-around-time* by enabling reuse of existing designs. As shown in the Figure 1-1, we would like to design parts of the chip at different places with the help of design tools that will enable collaboration and design integration across geographical boundaries.

In addition to distributed access to cell libraries, it is also important to have transparent access to design tools irrespective of the geographical location of the tool. The vision is a *plug-and-play* architecture of CAD tools, which will be available off-the-shelf and integrated in the designers environment. This will not only help reuse of tools, but will also facilitate the designer to utilize diverse expertise at tools level. The operating system or the hardware of the designer's computer should not prevent the designer from using a tool. In this context, interoperability of tools irrespective of the location and platform assume prime consideration.

With the availability of different enabling technologies and increased complexity of microsystem designs, the needs of VLSI system designers can be summarized as follows:

- Simplify the design process as seen by the user, which involves automating design flows and allowing the invocation of design tools without detailed knowledge of the internal tool operation.
- Access to design tools at all level of design abstraction from a simple terminal with simple and standard user interfaces.
- Transparent access to remote library modules and their specification in a distributed fashion.
- Sharing of information between users to reuse design flows and avoid duplication of design efforts.
- Enable exploration of design trade-offs by providing easy accessibility to new prototyping tools and technologies.

All these requirements motivated us to build a distributed framework where designers can access and utilize diverse resources from the desktop. The framework will allow designs tools to communicate with other in a standard integrated way and allow for other tools to utilize the capabilities of existing tools. The ultimate goal of a distributed CAD environment, is to allow the designer to have access to different tools and cell libraries in a transparent fashion, irrespective of the location and platform as shown in Figure 1-2.

1.2 The Web and the Internet

The advent of Internet has opened new vistas in the areas of distributed design and the World Wide Web has emerged as the most desirable platform for distributed access to information. But

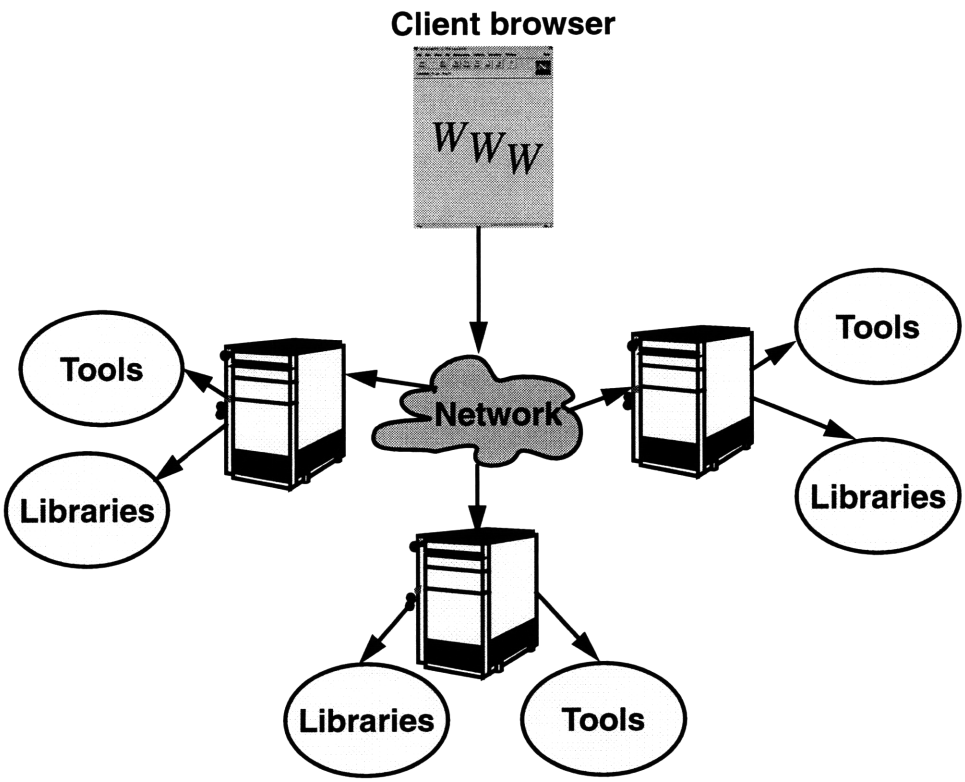

Figure 1-2: Distributed CAD Environment

there has not been equal efforts in making Web as a platform for distributed applications. The Web can also serve as a very attractive platform for distributed applications because of the following reasons:

- The Web is platform independent. The protocols and mechanism are very simple and do not reply on the underlying hardware.

- The Web's popularity is mainly its simple and standard browser interface. This means that many users are already familiar with a "friendly" user interface that Web has to offer.

- The limitations of only server side computation of the Web using form based CGI programming has been alleviated with the emergence of a platform independent programming language Java.

- The limitations of state-less connection are also overcome with Java as a full programming language being integrated in the Web browsers.

All these factors make the Web an attractive and universal platform for collaborative microsystem design.

1.3 Distributed Microsystem Design Framework

VLSI system design typically involves design specification, entry and storage, design verification, optimization and synthesis and generation of physical design. A distributed microsystem design framework will facilitate seamless access to cell libraries and VLSI CAD tools like synthesizers, generators and simulators distributed over the internet as shown in Figure 1-3). The complexity of the VLSI design process even in a single module or domain necessitates diverse skill sets at various levels of the process which mandates a distributed design methodology.

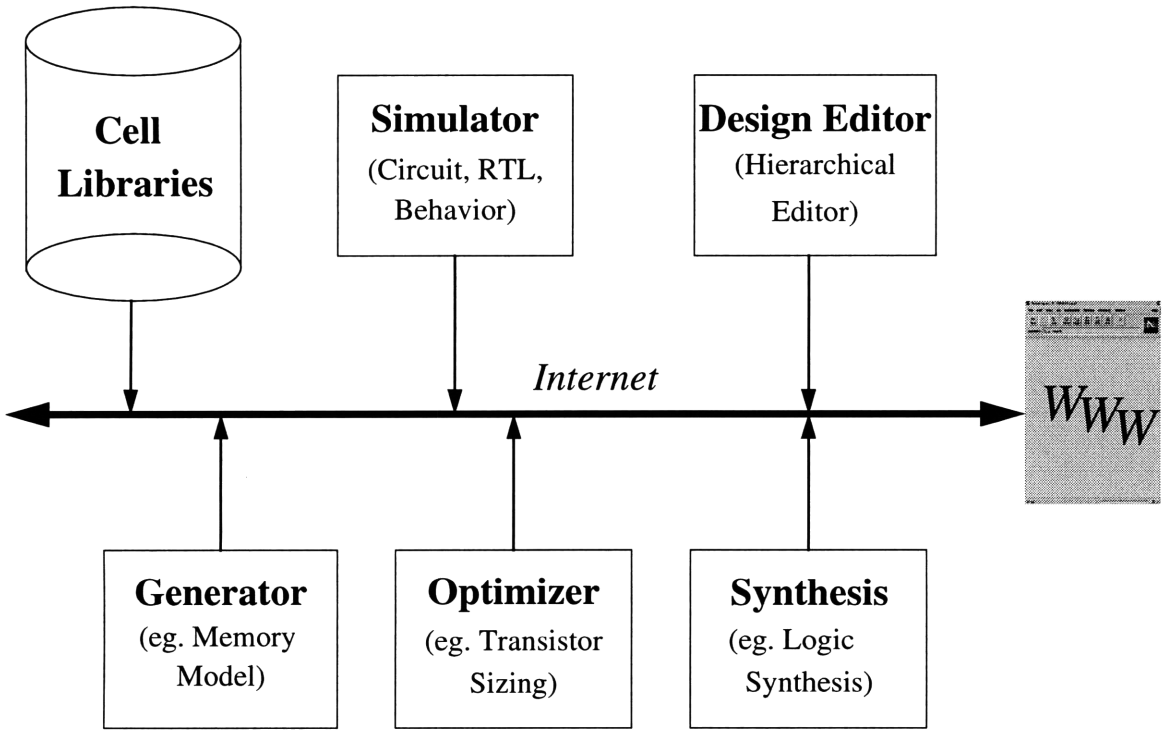

Figure 1-3: Distributed Microsystem Design Framework

The entry point in the framework is a hierarchical design editor which should be capable of accessing cell libraries located in different servers on the internet. This can be accomplished by two schemes. Cells can be stored in the Web Servers and are accessed as URL's from the Web servers or gateways can be used to retrieve the cells from distributed cell databases. The editor

should be capable of extracting the design into netlists of various formats. The editor should also have transparent interfaces with different CAD tools distributed over the Web. The Web tools can themselves be distributed in the sense each may invoke many other distributed Web tools. In the distributed framework, it should be possible to utilize the tools available on the Web to build new tools.

1.4 The Model of Web based CAD

Web based CAD poses to define a new era of electronic design in which tools, models and cell libraries are accessible from any remote location. Web based CAD enables the global vision of a distributed CAD framework described in the previous section. VLSI design involves many steps which may need intensive computation (e.g., simulation, verification, etc) and steps which do not need extensive computation, but requires complex user interaction (e.g. design entry). Therefore, a Web based CAD framework should partition the client-side and server-side jobs efficiently as shown in Figure 1-4, and all these tools should be seamlessly integrated in the framework shown in Figure 1-3.

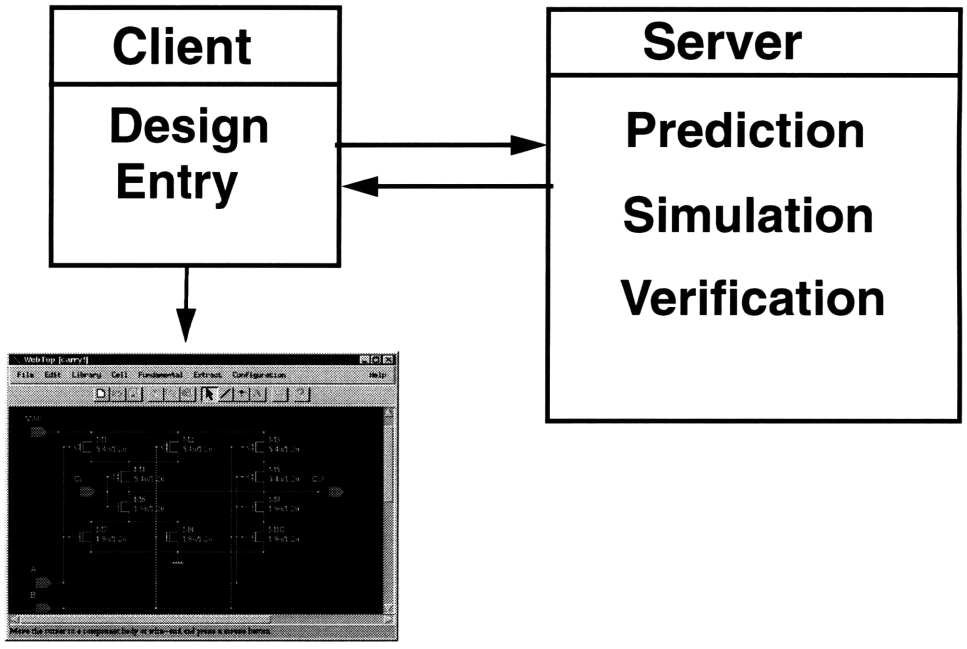

Figure 1-4: Partitioning Web Tools as Clients and Servers

Presently, VLSI designers usually use design tools and cell libraries located in the local file sys-

tem of their workstation. In our model of Web based CAD, the design process changes significantly. We simply use our favorite Web browser on our desktop or workstation, which has access to the internet to load a Web page in the browser, which fires up a design editor. The editor is used to create design cells and store them either locally or remotely in a cell database. Cells can be made available to others designers, by providing Web access to the design cells. Designs can also include cells transparently download from globally accessible libraries on the Web. When we have a design with possibly different cells loaded from multitude of locations, the netlister of the editor is used to netlist the design in a particular format. The editor will have interfaces to different Web based tools, which are used to perform simulation or estimation on the design netlist. Thus we are able to completely design a chip with tens of millions of transistor from a *simple desktop*, from anywhere in the world, utilizing the wide variety of tools available to us. We make use of the already existing expertise and skills in terms of cells and tools from different places without having to physically access the tools or models.

Let us take a closer look at the various aspects which are fundamentally different from the way VLSI design is done today.

- **No Installation of CAD tools:** Web based CAD does not require any client end installation of tools. Tools are either downloaded or run at the server side. The user should be able to invoke the tools only by knowing some kind of reference or the location of the tool.

- **No Maintenance:** This is probably the most important change we will see in the way people use software CAD tools. A Web based CAD framework virtually reduces CAD tool maintenance cost for the client to zero. The tool vendor will manage the tool at their site and the user always gets to use the latest version of the software. Currently, a lot of effort of a designer goes into tool installation and maintenance in terms of version controls, license controls etc. Since all of the burden is on the tool vendor, the user is left only with use of the tool.

- **Access to Powerful Computation:** Many CAD tools require extensive computational power, which many small designer houses cannot individually afford to have. The server based computational aspect of the Web based CAD framework will allow users to make use of high performance computing provided by some vendors. This will allow high performance computing resources like supercomputers, network-of-computers to be available to a large user base, and give small design houses access to the computation resources that are becoming a

requirement for VLSI design.

- **Pay-per-use CAD:** Making tools available on the Web will allow tool vendors to use a *pay-per-use* model. A pricing technique like per-use will need methods that can guarantee delivery of service to users while keeping track of the count of actual use.

Irrespective of the business prospects, Web based CAD allows designers to access and experiment with a wide variety of options and which would foster a distributed collaborative design environment. In such a Web-based framework the tools should have the capability to exchange information efficiently such that the designers could make effective use of the wide variety of tools available to them.

1.5 Meta Web tools

In our vision of distributed Web based CAD, we have distributed tools which can be easily integrated in the framework. As stated in Section 1.3, it is also desirable that we use existing tools to build new tools. The tools themselves can be distributed and invoke other distributed tools internally. Eventually, we will have a framework where it will be easy to build new Web tools which exploit the capabilities of existing Web tools. We call these Web tools *Meta Web Tools.*

Let us consider the following example which demonstrates the usefulness of such a hierarchical framework. The increasing complexity and heterogeneity of current day systems makes early design tradeoffs and analysis a necessity. Typically such design exploration is performed at a time that specifications are incomplete or ill-defined, and no behavioral or structural descriptions have been formulated. In current day design practices, this level of design abstraction, which may be called the conceptual level, is addressed in an ad-hoc fashion without much support from tools or without fully using existing data and models.

In the conceptual level of VLSI design, early exploration of power dissipation of circuits has assumed paramount importance. With a multitude of different technologies and power estimation tools available as depicted in Figure 1-5, designers would like to estimate the power dissipation for different architectures and technologies. It is often desirable to specify their design in mixed levels of abstraction. This could be the case if some parts of the design are complete while other parts are not. For example, the designer may want to specify the design as a combination of high level description (C or VHDL program), parameterized library modules (RAM, ROM, adders, multipliers) and low-level schematics (transistors and gates). Several Web based tools exist to estimate power

dissipation at different levels of abstraction (e.g., Pythia - a transistor-level simulation based Verilog Power Estimation Tool at MIT [29], PowerPlay of UC, Berkeley [15], PPP - Power Estimation and Synthesis of Low-Power Circuits at Stanford [3]). Since all these tools are available on the Web and if there exists a transparent framework to integrate these tools, a designer at MIT can build a new tool, which given a mixed level circuit design (specified in various levels of abstraction) will generate the estimated power dissipation of that circuit. The new tool will call appropriate tools on the Web for the specific technology for appropriate abstraction of the design and then generate the total power dissipation of the circuit. Once this new tool is available on the Web, it could be used by other integrated CAD environments to estimate the power dissipation. Therefore we have a hierarchy of tools which encapsulates other tools on the Web to build a new tool on the Web. Such a hierarchical framework on the Web facilitates distributed design and efficient exchange of information. Figure 1-5 shows the distributed hierarchy of tools.

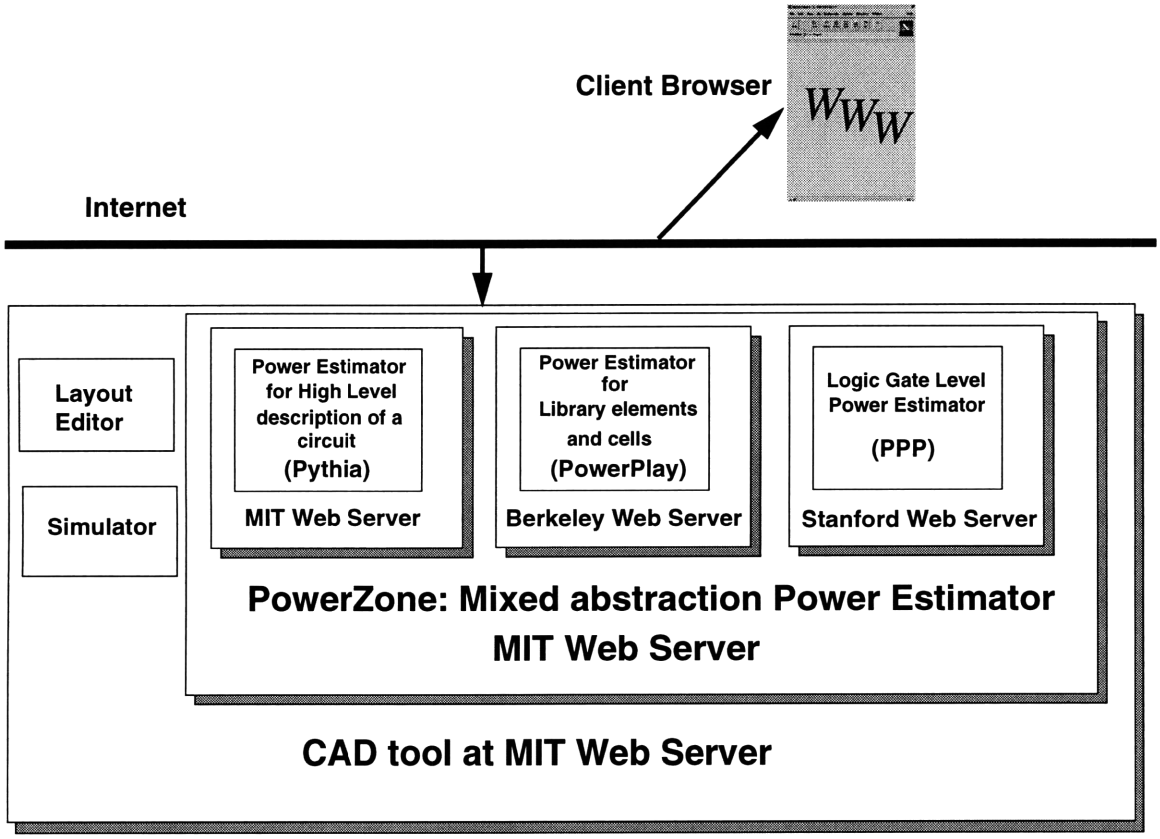

Figure 1-5: Hierarchical Web based design tools

1.6 Related Work

The popularity of the internet and the Web has lead to many distributed design efforts. In [2], Bentz et al. have proposed an information based design environment. They describe an environment which helps users collect and manage information in a uniform fashion, independent of the abstraction levels or implementation platforms.

In [15], Lidsky and Rabaey present a World Wide Web based prototype tool, PowerPlay which helps in system level exploration of power consumption. PowerPlay uses the HTML form-based user interface which prompts the user to select a subset of available library and their parameters. The design is then submitted to a script which calculates the power, area and timing informations of the design. The methodology combines pre-characterized and user defined modes to provide quick and "as accurate as possible" power estimation in a spread sheet like format at the earliest stages of design. PowerPlay uses a distributed data modeling environment for early exploration. Using a hyperlinked spreadsheet as a basis for exploration, the designer can incorporate any type of model, from SPICE simulations to simple equations, to perform their desired exploration. PowerPlay addresses the issues both on the environment required and the challenges of modeling at the conceptual level.

PPP - A Gate-Level Power Simulator, which provides a Web-based integrated environment for synthesis and simulation of low-power CMOS circuits has been presented in [3, 1]. The graphical interface of PPP is a dynamically generated tree of interactive HTML pages that allow the user to access and execute the tool by using a Web-browser.

There have been efforts to provide Web-based Interfaces to executable CAD/CAM softwares. Links to different Web based tools could be found at [17]. The Purdue University Network Computing Hub (PUNCH) [11], is a set of network-based laboratories that provide toolkits of programs for various fields. Users can access software, run the tools, view results, and download data via a specialized WWW-server that interfaces to standard WWW-browsers. The Hub is currently being used for on- and off-campus education, to support research activities, and as a testbed for network-based, on-demand high-performance computing. The VLSI Design Hub is a network-based simulation laboratory that provides a toolkit of programs for simulating semiconductor materials, processes, devices, and circuits.

The WADE (Web-based Automated Electronic Design Environment) [4] project intends to develop a prototype of a self-contained, automated CAD Computing Center accessible transparently to everyone on the Internet. Users can specify the target technology, design constraints and upload

their design files to the WADE Computer Center. Software at the Computer Center can apply parallel and distributed computing resources automatically to the CAD task, producing a circuit implementation as quickly as possible. The resulting circuits can then be returned back to the user. In the WADE framework, users authenticate themselves, upload their design files to the Compute Center and receive the results back.

Although, the previously mentioned tools and frameworks provide access to remote computing resources, they do not discuss a framework where tools can be transparently invoked from other tools, nor do they provide a framework which integrates client-side and server-side computation.

The explosive growth of the World Wide Web has lead to a need for support for distributed, collaborative and scalable applications. WebOS [25] (Software Support for Scalable Internet Services) are focusing on the infrastructure necessary to build distributed Web application. In [22], Vahdat et al. demonstrate a different vision of the Web operating environment where computation and storage servers in the Web function cooperatively to provide efficient access to resources supporting Web applications. In [23], Vahdat et al. describe WebFS, a global file system layer allowing unmodified applications to read and write to the URL name space.

There has been significant efforts to exploit the Web and the internet as a platform for collaboration. The Collaborative Benchmarking Laboratory [5], intends to support a series of international and collaborative experiments by prototyping Internet-based collaborative and reconfigurable workflows. In [14], Lavana et al. use executable directed hypergraphs to describe collaborative design activities on the internet. The directed hypergraph model supports workflow composition and reconfiguration while accessing and executing programs data and computing resources across the internet, synchronous and asynchronous peer-to-peer interaction between members of any team during workflow composition and execution. The collaborative workflow is based on a reconfigurable and composable graph-based representation of objects, such as programs files, decisions scripts, host definitions, team definitions and recorded sessions which are shared across the Internet. Workflows can be executed and controlled in real-time by several distributed participants or can be scheduled for batch execution and interaction.

Reuben [12], a user-reconfigurable distributed workflow environment prototyped in Tcl/Tk whose goal is to introduce and to support collaborative peer-reviewed benchmarking experiments in EDA (Electronic Design Automation) on the Internet, is also available. Reuben permits workflow composition and reconfiguration while accessing and executing programs, data, and computing resources the internet. It allows real-time communication between among team members and permits

interaction between any team member and any object in the workflow. Users can view programs remotely, and Reuben can play back and record transactions.

OmniDesk [13], which has been implemented as a Tcl applet, creates a user-configurable desktop within the web browser window. User can place objects onto the OmniDesk ranging from windows that display the contents of a directory or a file on a remote host to OmniFlow applets that can execute any sequence of user-defined and data-dependent tasks. An OmniFlow is a user-created directed dependency graph of data program decision and OmniFlow nodes, where data and program nodes may reside anywhere on the Internet. The OmniDesk and OmniFlows present a good generic workflow based collaborative framework, but there has not been much effort to integrate tools in a transparent fashion, independent of the technology (Objects and CGI) and platform. Moreover, the framework does not discuss the issues of tool specification, which would allow tools to invoke other tools. The problems of the "applet security" are also similar to that of Java applets. Java as an internet language has major strengths over Tcl, and also provides the object-oriented benefits such as remote method invocations, which provides transparent access to distributed computation.

The WELD project at UC Berkeley [27], aims to construct a distributed CAD design environment enabling Internet wide IC design for the electronic industry. The WELD group has developed a Java Object Database Server supporting persistent object class management. They are also looking into the Java client side infrastructure to enable a distributed collaborative design environment. At the 1996 Design Automation Conference (DAC), the WELD group demonstrated a prototype of a Web-based CAD environment. The WELD project addresses most of the issues regarding Web based CAD and is very similar to the goals and objectives as stated in Section 1.3, and it will be interesting to compare both the approaches. The main difference of the WELD approach from our framework is the distributed architecture. In our framework, we try to integrate different Object-Web technologies to deliver distributed CAD tools and design libraries on the Web. We simplify the architecture to a large extent by providing an open ended three-tier Object-Web architecture which will be discussed later in the thesis. In the WELD architecture, the middle tier consists of network services, like the distributed data manager, proxies and registry services, where as in our approach the middle tier solely comprises of the middleware which will interface the clients to the application and data tier.

We have seen that there are numerous instances to design tools being available over the Web, but there have been limited attention to design a framework which would allow us to utilize the different tools over the Web in a transparent fashion. In this thesis we show how a distributed

framework could be built over the Web, utilizing the core Web technologies, to support efficient communication and data exchange between different Web based tools. We will also present WebTop, a Java schematic editor, which interfaces to distributed Web tools and design libraries.

Chapter 2

Web based Technologies

"The Web is about knowledge exchange, not just broadcasting", – Dan Connolly, W3C Architecture Domain Leader

In this chapter, we describe the technologies, mechanisms and protocols available that support distributed applications with the standard Web interface. Current tools support execution of applications at the server side using HTTP and CGI and use Java technology, scripts and plug-ins to create applications that are executed locally on the user's machine. Another approach is to use object technologies to define, locate and request computational services from participating applications, both remotely and locally. In this approach, the Web takes the role of providing uniform access and presentation mechanisms.

2.1 The Hypertext Transport Protocol

The Hypertext Transport Protocol (HTTP) is the principal means by which a Web server and client communicate with each other. Under this protocol, a client sends a *request* to the server to retrieve a document or execute a script. The server compiles and sends back a *response* containing the requested output or an error message if something went wrong. Together, the request and the response form a *transaction*, a single interaction between the server and the client. The protocol is basically stateless, a transaction consisting of a *connection, request, response* and a *disconnection* (Figure 2-1). A feature of HTTP is the negotiation of data representation, allowing systems to be built independently of the development of new advanced representations.

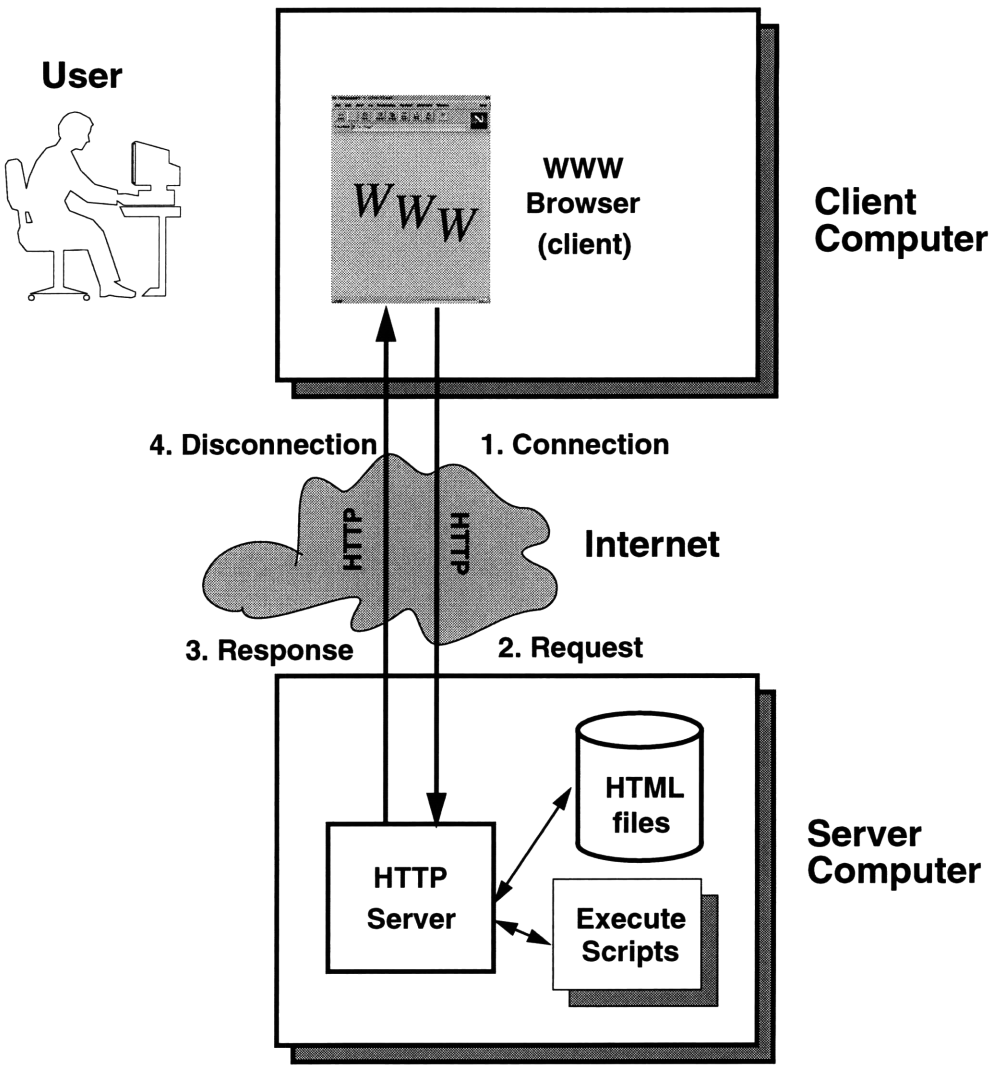

Figure 2-1: The components of a simple WWW interaction

2.1.1 Connection

The client makes a TCP-IP connection to the host using the domain name or IP number, and the port number given in the address. If the port number is not specified, 80 is always assumed for HTTP. HTTP runs over TCP, but could run over any connection-oriented service.

2.1.2 Request

A request is nothing more than a message from a client to a server. It consists of a line of ASCII characters terminated by a CRLF (carriage return, line feed) pair, optionally followed by any

number of header lines. A blank line consisting only of CRLF signals the end of the headers, which may be followed by a message body.

There are two common methods or syntax of the request line (the first line), GET and POST methods. GET method is usually used for simple document retrieval, database lookup and similar operations. This request consists of the word "GET", a space, the document address, (omitting the "http:"), host and port parts when they are the coordinates used to make the connection. The document address will consist of a single word (i.e. no spaces). If any further words are found on the request line, they MUST either be ignored, or else treated according to the full HTTP specification. The final portion specifies the version of the protocol in use, namely HTTP/1.0 or HTTP/1.1. The headers that accompany an HTTP request convey additional information about the connection and a description of the headers can be found in the HTTP specification [9]. A request for the home page, sent once the client has made a connection to the server `http:// apsara.mit.edu` would look like:

```
GET /home/index.html HTTP/1.0
Referer: http://apsara.mit.edu/index.html
User-Agent: Mozilla/1.22 (Windows; I; 32bit)
Accept: */*
Accept: image/gif
Accept: image/jpeg
Accept: image/x-xbitmap
```

We note that <CRLF> character separates each line of the request. The CRLF is "\r\n" in Unix. The request is terminated with a line containing only CRLF.

POST method is useful when the client wants to send a relatively large amount of data to the server. The POST method uses the method body to send the additional information from the user rather than encoding it in the part of URL as in GET. The client can send anything in the body as long as the server and the script can understand it. Different encodings of the data can be used. The default HTML encoding is `application/x-www-form-urlencoded` which encodes the data as ampersand-limited name/value pairs. File uploads with the HTML tag `<input type=file>`, uses the encoding of `multipart/form-data`. A client request with POST method and encoding type of `application/x-www-form-urlencoded` will typically look like:

```
POST /cgi-bin/test.cgi HTTP/1.0
Referer: http://apsara.mit.edu/index.html
User-Agent: Mozilla/1.22 (Windows; I; 32bit)
Accept: */*
Accept: image/gif
Accept: image/jpeg
Accept: image/x-xbitmap
Content-type: application/x-www-form-urlencoded
Content-length: 24

first=Debashis&last=Saha
```

2.1.3 Response

The HTTP response consists of a status line which, like the HTTP request is then followed by several header lines, a blank line and possibly a message body. The response to a simple GET request is a message in Hypertext Mark-up Language (HTML) [8]. This is a byte stream of ASCII characters. The first line of the server response begins with a protocol identifier, followed by the status code. Additional information may follow, in the format of a MIME message body. The significance of the data depends on the status code. The Content-Type used for the data may be any Content-Type which the client has expressed its ability to accept, or text/plain, or text/html. That is, one can always assume that the client can handle text/plain and text/html. The following listing shows an example response to the GET request to home page at `http://apsara.mit.edu/home/`.

```
HTTP/1.1 200 OK
Date: Thu, 30 Oct 1997 18:18:13 GMT
Server: Apache/1.2b7
Connection: close
Content-Type: text/html
Last-Modified: Wed, 20 Aug 1997 23:10:44 GMT
ETag: "28920-d1d-33fb7974"
Content-Length: 317
Accept-Ranges: bytes
```

```
<HTML>
<HEAD>
<Title> Home Page of Debashis Saha </Title>
<meta name="keywords" content="Debashis Saha, Home page of Debashis Saha">
</Head>

<body TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000FF"
ALINK="#85BAE5" VLINK="#5875E5">
Hello World!
<a href="mailto:debashis@mit.edu"> debashis@mit.edu </a>
</body>
</HTML>
```

Error responses are supplied in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

2.1.4 Disconnection

The TCP-IP connection is broken by the server when the whole document has been transferred. The client may abort the transfer by breaking the connection before this, in which case the server shall not record any error condition.

2.2 The Common Gateway Interface (CGI)

The Common Gateway Interface is a mechanism to present dynamically generated information on the World Wide Web. A Hypertext Transport Protocol (HTTP) server is often used as a gateway to a legacy information system: for example, an existing body of documents or an existing database application. The Common Gateway Interface (CGI) is an agreement between HTTP server implementors about how to integrate such gateway scripts and programs. CGI programs are usually referred as scripts which run on the server machine and produce the output (usually HTML output) to be displayed on the clients browser.

CGI is the part of the Web server that can communicate with other programs running on the server. With CGI, the Web server can call up a program, while passing user specific data to that program. The program processes the data and passes the program's response back to the

server, which sends the data back to the client. CGI turns the Web from a simple collection of static hypermedia documents into an interactive medium, in which users can ask questions and run applications.

The request for a CGI program looks the same as it does for all Web documents. The difference is that when a server recognizes that the address being requested is a CGI program, the server does not return the file content, instead it tries to execute the program.

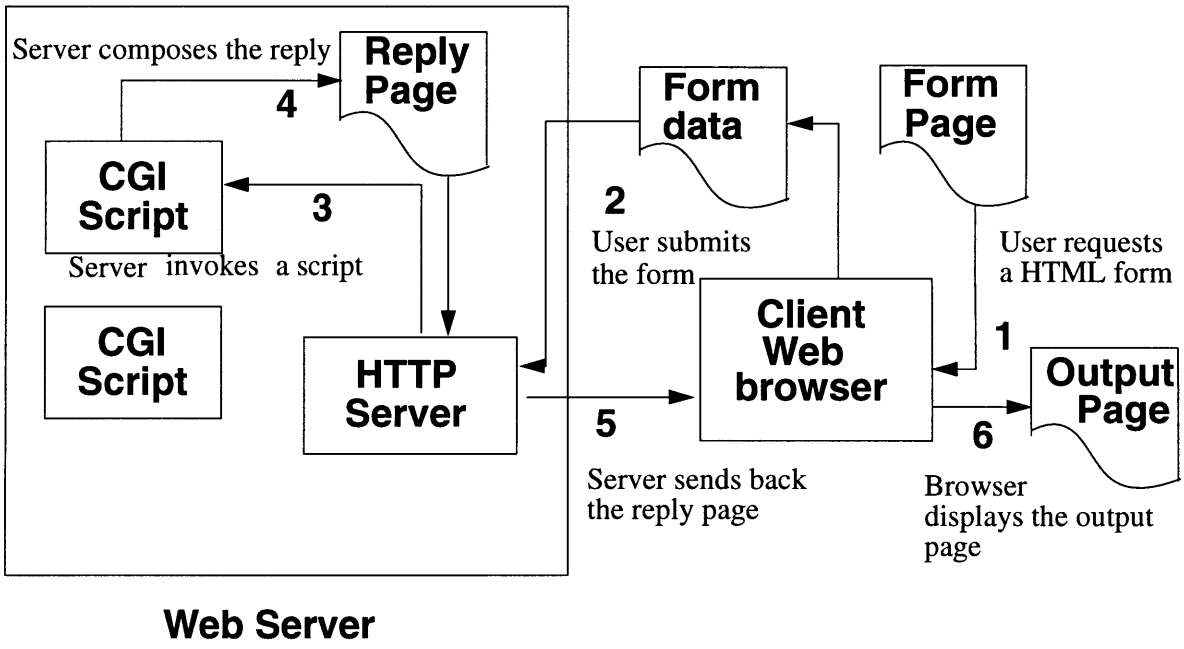

Figure 2-2: Data Flow during execution of a CGI Script

The following sequence of events, as shown in Figure 2-2, occur during the execution of a CGI script:

1. The user uses an HTML form to enter the data to be processed by the CGI script. Forms are HTML pages that make use of special tags to provide an interface to solicit information from the user.

2. After the user enters the information, he submits the form to the Web server where the CGI script to be executed resides. The address of the CGI script is identified from the `action` tag of the form.

3. The server recognizes that the requested address is a script and therefore executes the program. The way that CGI programs get their input depends on the native operating system.

On a UNIX system, CGI programs get their input from the standard input (STDIN) and the form environment variables, which the server sets before executing the script. These variables store such information as the format of the input, the length of the input (in bytes), the remote host and the user passing the input and other client information. They also store the server name, the communication protocol and the name of the software running the server.

4. The CGI script then produces the reply. The CGI program can create and output a new document or provide the URL of an existing document. On UNIX, the programs send their output to standard output (STDOUT) as a data stream. The data stream consists of two parts. The first part is either a full or a partial HTTP header that at minimum describes the format of the returned data (e.g., HTML or plain text etc.) and a blank line signifies the end of the header section. The second part is the body, which contains the data conforming to the format of the type reflected in the header. The server reads the data output from the standard output and composes the reply page. If the CGI script outputs a complete header, the server does not modify the header. If the header was a partial one, the server is responsible for adding the complete header information.

5. The server then returns the reply page to the client using the HTTP protocol.

6. The client browser displays the output page. Thus, the processed data from the CGI script gets displayed on the client browser.

For example the following request might be send to the server requesting the home page of `Debashis Saha`.

```
POST /cgi-bin/give-me-home-page.pl HTTP/1.0
User-Agent: Mozilla/1.22 (Windows; I; 32bit)
Content-type: application/x-www-form-urlencoded
Content-length: 24

first=Debashis&last=Saha
```

As a result of execution of `give-me-home-page.pl`, the following page may be generated by the script:

```
Content-Type: text/html
```

```
<HTML>
<HEAD>
<Title> Home Page of Debashis Saha </Title>
</Head>

<body TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#0000FF"
ALINK="#85BAE5" VLINK="#5875E5">
  Hello World!
<a href="mailto:debashis@mit.edu"> debashis@mit.edu </a>
. . . . . .
</body>
</HTML>
```

The server then adds the complete header info and sends it back to the client.

CGI scripts provide a very useful mechanism to provide remote access to CAD tools. Since many CAD tools take inputs which are either files or some parameters in plain text, HTML forms can easily be used to capture the inputs from a browser interface. For example, a power estimation tool could have simple a Web interface using the CGI mechanism. In Chapter 3, we will present examples of CGI based CAD tools and will also demonstrate how CGI tools could be used to build hierarchical CAD tools.

2.3 Java

CGI programs and forms lack the interactivity and complex user interface. No designer would be satisfied with the limited display of a CGI based program with a HTML output. Designers need to modify and view different designs in real time. Java allows us to do complex client-side processing in a platform independent manner.

Java is a full object-oriented programming language and comes with a rich set of Application Programmers Interface (APIs), and has been widely accepted as the standard for Web computing, as shown in Figure 2-3. Java source code is compiled to byte-codes whose target architecture is a Java Virtual Machine (JVM). The Virtual Machine is embeddable within other environments, like the web-browser and operating systems. A class loader can load classes over the network and a

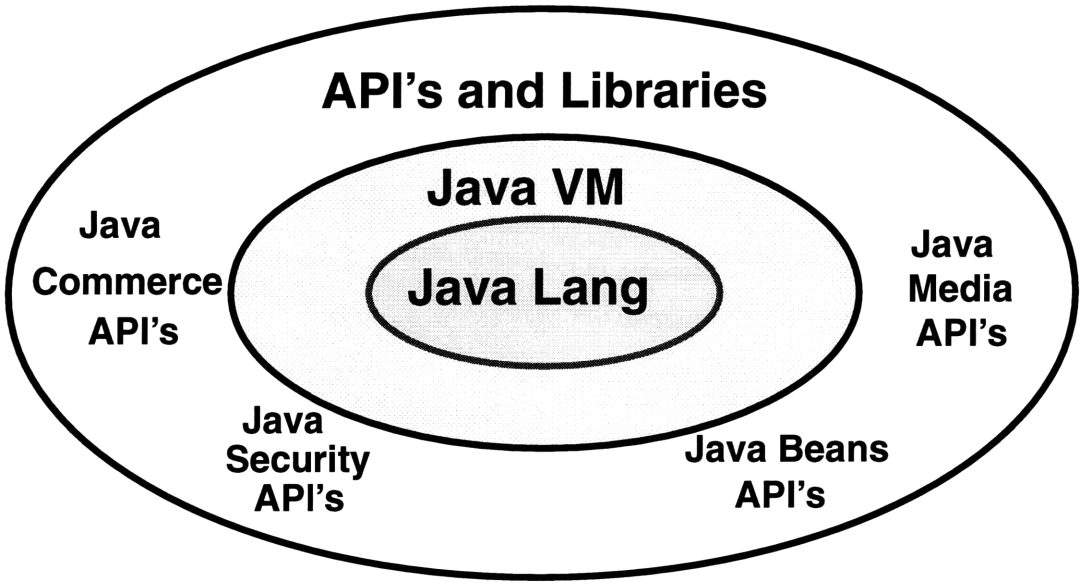

Figure 2-3: JAVA: More than a language

byte-code verifier verifies the byte-codes while reading them. Java has been designed such that a compiled Java class can be conveniently handled with ordinary text and graphics. Java *applets* can be embedded in HTML pages, which are loaded from the Web server. The browser interprets the Java byte-codes and run them as a mini-application on the client side.

Since Java is a true programming language, it is much easier to make an applet interactive than to do the same with a Web page. Complex user interfaces using the mouse to draw and edit images, which are not possible with HTML and CGI can be easily incorporated in a Java applet. Applets use modern graphical user interfaces (GUI) like text boxes, buttons, list boxes etc. With Java, the processing is off-loaded to the user's system and thus enable client side processing without the need to go across the network for every small application. Moreover, the problems of the non-persistent connection of HTTP and CGI are alleviated since Java applets run as an independent application.

The Virtual Machine refers to the active interpreter which executes Java byte-codes. Since the Virtual Machine does not necessarily correspond to any particular hardware or operating system, the Java class files are portable to any implementation of the Virtual Machine. This is the essence of Java's portability. Java's portability enables the dynamic content of the servers Web site to be accessible and completely functional to the remote users. Therefore applications which do not need extensive computational power could be handled well with Java based interface. But leaving all the work to Java based client-side processing would hinder our attempts to do CAD with a

simple desktop. Therefore what we need is a partitioning of an application into Java based client processing and traditional remote server-side computation.

In summary, Java enables a philosophy of *" Write once, Run anywhere"* for programs. The features which make Java so attractive are:

- Java is Object-oriented.
- Java is a cross platform language enabling easy portability.
- Java inherently supports multi-threading.
- Java has API's for transparent network access and can be inlined in a HTML document in a Java-capable browser.

2.4 Distributed Object Technologies

Distributed computing enables the development of distributed applications across heterogeneous systems. With the growth of internet and the advances in networking, distributed computing offers a vision of flexible computing with no boundaries. Distributed computing has gained a new direction with a vision of distributed object-oriented computing in which, there is no essential distinction between objects that share an address space and objects that are on two different machines with different architectures and different geographical location. Distributed computing in this context refers to programs that make calls to other address space, possibly on other machine. Objects can therefore invoke a method of another object which may be remote to the calling object. In such a system, an object, whether remote or local, is defined in terms of a set of interfaces. The implementation of the object is independent of the interface and hidden from other objects. This vision can be seen as an extension to the remote procedure call (RPC) to the object-oriented paradigm.

In this section we describe in brief the two most popular distributed object technologies, CORBA and the Remote Method Invocation of Java.

2.4.1 CORBA

This section gives a broad overview of the Object Management Architecture (OMA) and its object communication mechanism, CORBA [18]. These standards are published by the Object Management Group (OMG). The Object Management Group is a non-profit consortium consisting of over

750 software vendors and members that is dedicated to promoting the theory and practice of object technology (OT) for the development of distributed computing systems. Their goal is to provide a common architectural framework for object-oriented applications based on widely available interface specifications.

The Object Management Architecture is the architectural framework introduced by the Object Management Group with a view to drive the industry towards interoperable, reusable, portable software components based on open, standard object-oriented interfaces.

Object Management Architecture

The OMA Guide has two components: The Core Object Model and the OMA Reference Architecture.

The Core Object Model defines all the object-oriented concepts on which the CORBA specifications are built. It defines an object model and a framework for extending the model. The main concepts defined in the Core Object Model are objects, operations, non-object types, interfaces and substitutability.

The OMA Reference Architecture, shown in Figure 2-4, describes a framework for distributed application integration. The architecture contains:

- **The ORB:** The Object Request Broker (ORB) is a message bus for communicating requests to invoke operations on objects, irrespective of any object's location or implementation. Compliance with the Object Request Broker standard guarantees portability and interoperability of objects over a network of heterogeneous systems.

- **Object Services:** Object Services standardize the life-cycle management of objects, like Naming, Event Notification, Transaction Management, Trading etc. Interfaces are provided to create objects, to control access to objects, to keep track of relocated objects, and to control the relationship between styles of objects.

- **Common Facilities:** Common Facilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. These are set of higher-level object specifications providing commonly required services to applications, such as printing, document management, database, and electronic mail facilities.

- **Domain Interfaces:** Domain Interfaces represent vertical areas that provide functionality of direct interest to end-users in particular application domains. They are designed to perform

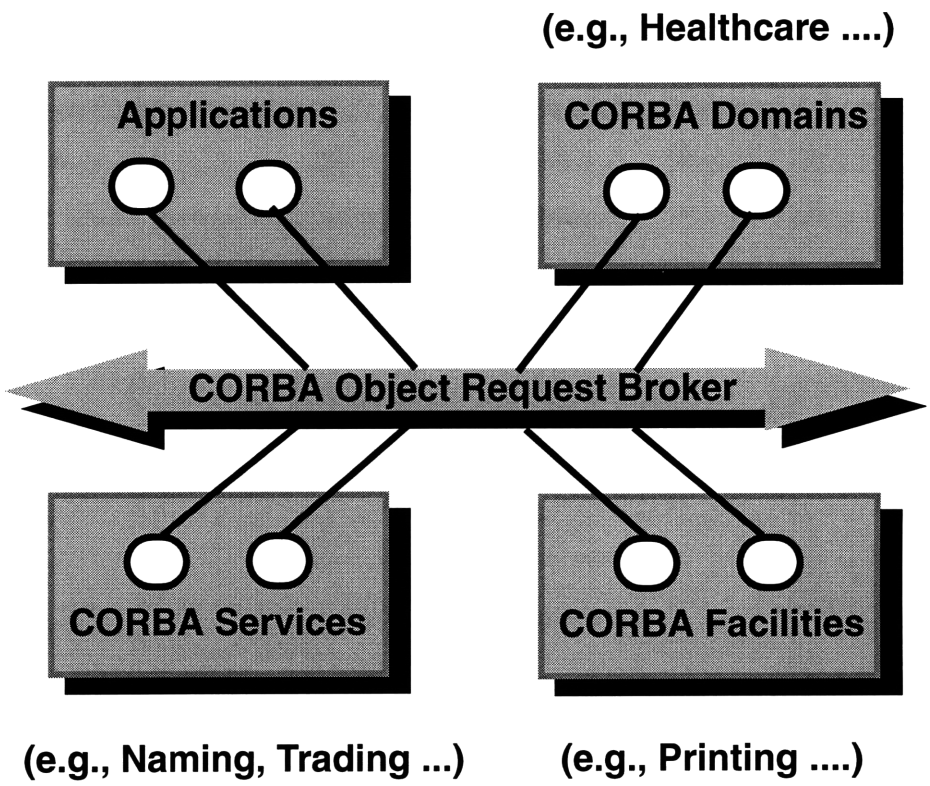

Figure 2-4: The OMA Reference Architecture

particular tasks for users within a certain vertical market or industry.

- **Application Objects:** The Application Interfaces represent component-based applications performing particular tasks for a user.

CORBA Specification

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by (OMG) along with the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an ORB. CORBA 2.0 enables client/server object interaction and interoperability by specifying how Object Request Brokers (ORB) from different vendors can interoperate.

The ORB is the middleware that establishes the client-server relationships between objects.

The ORB provides *location transparency* and *programming language transparency.* Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems. Figure 2-5 shows the basic mechanism of communication using an ORB.

In fielding typical client/server applications, developers use their own design or a recognized standard to define the protocol to be used between the devices. Protocol definition depends on the implementation language, network transport and a dozen other factors. ORBs simplify this process. With an ORB, the protocol is defined through the application interfaces via a single implementation language-independent specification, the IDL. And ORBs provide flexibility. They let programmers choose the most appropriate operating system, execution environment and even programming language to use for each component of a system under construction. More importantly, they allow the integration of existing components. In an ORB-based solution, developers simply model the legacy component using the same IDL they use for creating new objects, then write *wrapper* code that translates between the standardized bus and the legacy interfaces.

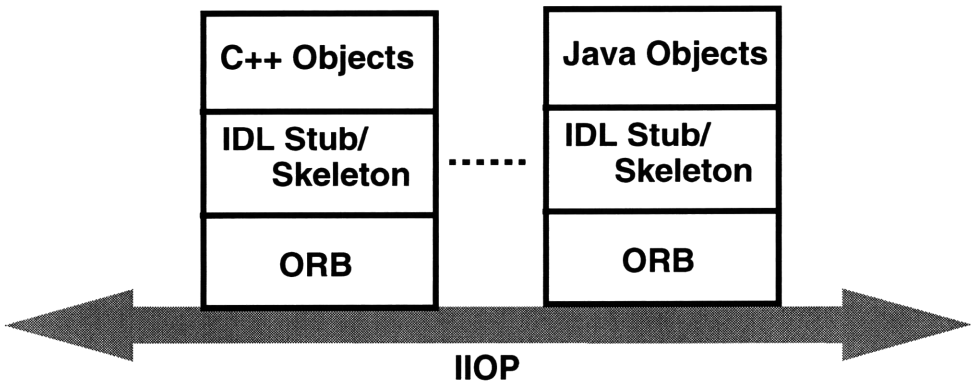

Figure 2-5: The CORBA Communication mechanism

CORBA is a signal step on the road to object-oriented standardization and interoperability. With CORBA, users gain access to information transparently, without them having to know what

software or hardware platform it resides on or where it is located on an enterprises' network. The communications heart of object-oriented systems, CORBA brings true interoperability to today's computing environment. With the recent popularity of the Internet, the future of CORBA lies greatly in its use in open distributed tools and and services

ORB Structure

We briefly describe the ORB structure, shown in Figure 2-6.

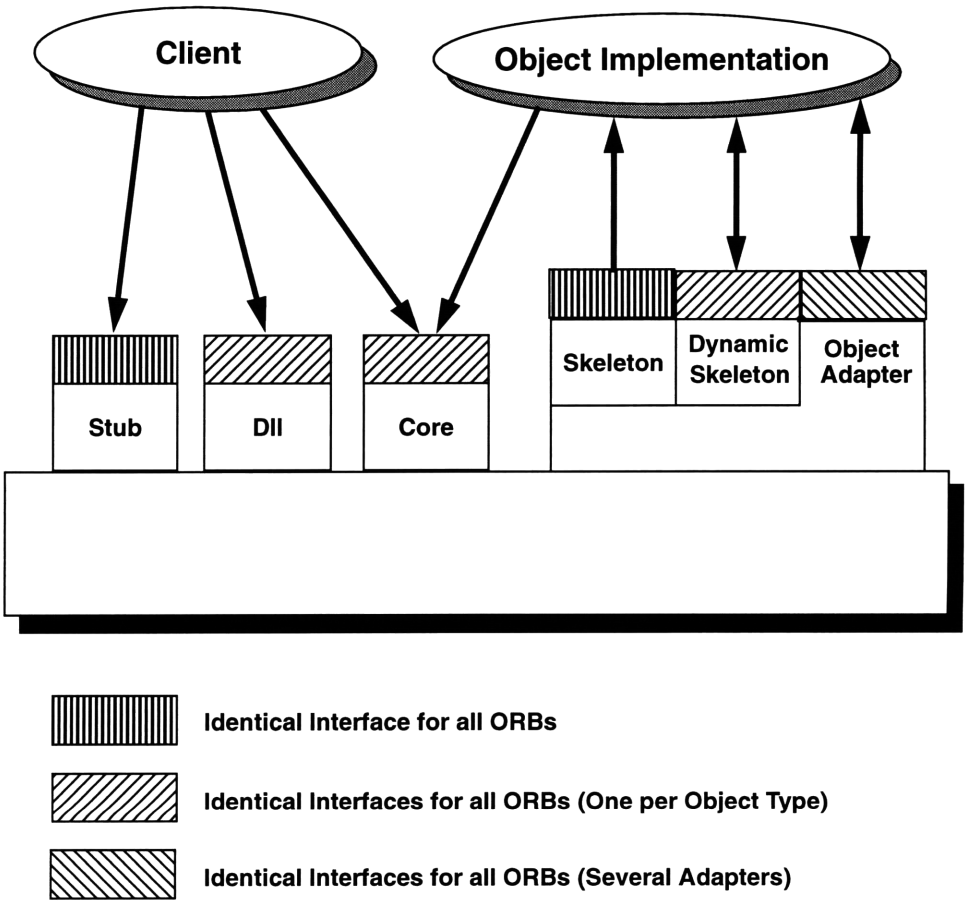

Figure 2-6: The Structure of the Object Request Broker Interface [18]

The ORB is implemented in several parts:

- *IDL compiler generated code:*
  - Stub Code, which is linked to the CORBA client
  - Skeleton Code, which is linked to the CORBA object implementation.

- *An ORB agent*, which locates and launches servers and facilitates client communication with servers.
- *Library code*, which is used by the above components.
- *IDL stubs:* The code generated for a specific IDL interface to allow static invocation of operations on that interface via proxy objects.
- *Dynamic Invocation Interface (DII):* This is a way to invoke operations without the IDL stubs.
- ORB Interface: An interface offering miscellaneous services from the ORBs to clients and servers.
- *IDL Skeleton:* The code generated for a specific IDL interface that invokes object implementations of that type.
- *Dynamic Skeleton Interface (DSI):* A generic interface that allows interpretation of incoming requests to a server for IDL types that were not known at compile time.
- *Object Adapter:* Because the ORB Core is free to be implemented in a variety of ways, adapter interfaces are defined that provide standard interfaces to servers. The Basic Object Adapter (BOA) of the CORBA 2.0 specification, is a component of the ORB that is capable of activating servers whose objects are required by invocations. After a server is ready, it must inform the BOA that it is ready to receive incoming requests.

2.4.2 Java RMI

Java Remote Method Invocation (RMI) enables the programmer to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. A Java program can make a call on a remote object once it obtains a reference to the remote object, either by looking up the remote object in the bootstrap naming service provided by RMI or by receiving the reference as an argument or a return value. A client can call a remote object in a server, and that server can also be a client of other remote objects.

Java Remote Method Invocation (RMI) is an interprocess protocol for Java, allowing Java objects living in different Java Virtual Machines to invoke transparently each other's methods.

Since these Virtual Machines can be running on different computers anywhere on the network, RMI enables object-oriented distributed computing.

RMI provides a simple and direct model for distributed computation with Java objects. These objects can be new Java objects, or can be simple Java wrappers around an existing API. Java embraces the "Write Once, Run Anywhere model". RMI extends the Java model to be run everywhere.

Java RMI provides the Java programmers with an efficient, transparent communication mechanism that frees them of all the application-level protocols necessary to encode and decode messages for data exchange.

Remote objects written in Java RMI interact approximatively in much the same way CORBA objects do:

- server objects publish their interfaces to make them available to RMI clients.
- stub classes deal with binding to the remote objects and do the client-side data marshaling.
- skeleton classes on the server-side handle incoming calls.

The communication mechanism is shown in Figure 2-7 . Without any surprise, this looks very much like the elementary ORB structure of Figure 2-5 .

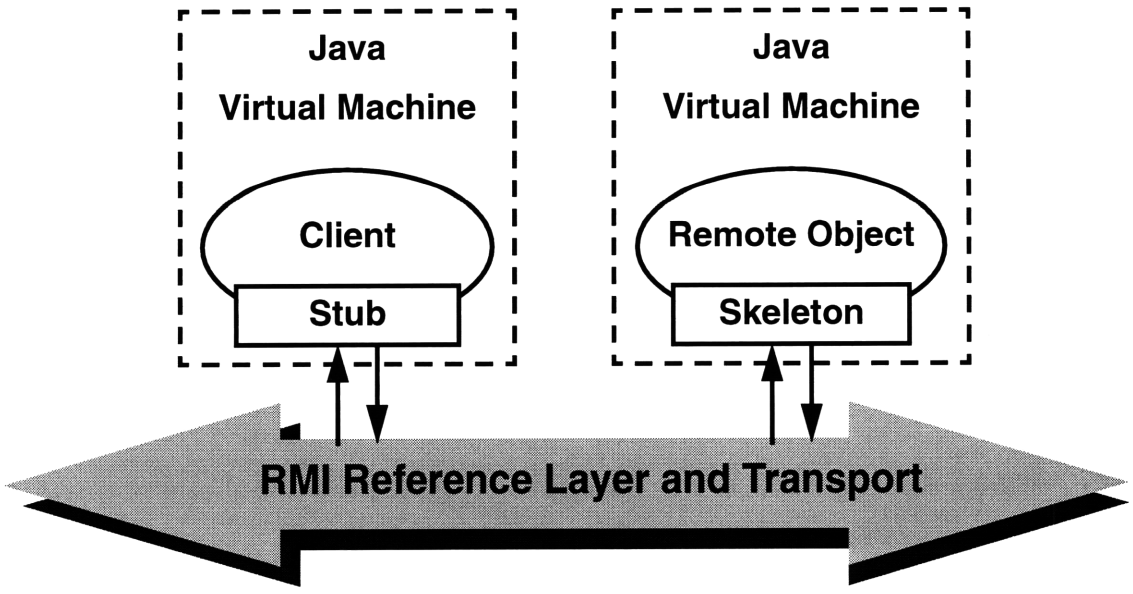

Figure 2-7: The RMI Mechanism

Following are some of the new features or divergences of Java RMI specific to JDK1.1 implementation.

- The results of a remote invocation, as well as the arguments that came along with the initial request, are passed by value rather than reference. This is because object references are only useful within the same Virtual Machine. Instead, ordinary objects are passed by value, by copy, between two different Virtual Machines. To do this copy operation, RMI uses the new object serialization service, which flattens a local Java object's state into a serial stream that it can then pass as a parameter inside a message.
- A Remote object, an object that implements the java.rmi.Remote interface, can nevertheless be passed by reference and not by copying the actual implementation.
- Clients of remote objects interact with remote interfaces, never with the implementation classes of those interfaces.
- Clients that invokes methods on remote objects have to deal with additional interfaces and classes, but also with a whole new set of exceptions.
- Extra security mechanisms are introduced to reinforce the control of the behavior of Java classes, and most of all stubs classes. If a SecurityManager is not explicitly installed by server objects, no remote invocation will be possible.

2.5 Summary

In this chapter, we presented the enabling technologies for Web based CAD. HTTP and CGI include protocols and mechanism, which provide access to tools running on the Web server. Java applets can be embedded in Web browsers to enable complex, platform independent client side processing. And finally, the distributed object technologies like CORBA and RMI enable transparent, distributed method invocation on remote objects.

In Chapter 3, we will present an Object-Web architecture for distributed VLSI design which will harness the strengths of both the Web infrastructure (HTTP and CGI), and the distributed object technologies (Java RMI and CORBA). We will also present a standard specification of CGI tools, which will help us to build a hierarchical framework for Web based CAD. We will present some examples of CGI based back-end tools for power estimation and simulation and show examples of hierarchical CGI Web tools using the framework.

In Chapter 4, we present a Distributed Microsystem Design Center, which includes WebTop, a Java hierarchical schematic editor and its Web oriented features. WebTop's Web based features include distributed cell access and storage, and its transparent integration with distributed tools. We will show how WebTop interfaces with CGI based simulation and estimation tools on the Web, and also how it uses which uses RMI mechanism for tool invocation on a *buffer generator*, which generates a driver cell for switching power supply. We will also present an experimental integration of PowerPlay, a system level power estimation tool developed at UC Berkeley, with WebTop.

In Chapter 5, we present some design examples, which include a 24 bit adder and an Inverse Discrete Cosine Transform (IDCT) chip, using WebTop and the associated tools of the Distributed Design Center. The chapter provides a proof of concept of the Web based design methodology. Chapter presents conclusions and directions for future research.

Chapter 3

Framework for Distributed Web-based CAD

"The Web and Corba will merge to provide the next infrastructure for distributed computing."

3.1 Object-Web Architecture

The rapid growth and acceptance of the World Wide Web has happened over the same time period in which distributed object systems, particularly the CORBA architecture, have stabilized and matured. CORBA is seen as the foundation for platform independent and interoperable information systems and the Web stands to deliver the services globally, using a widely accepted standard interface. The Web can become a predominant platform for software development, when the object technologies are combined with the Web based object oriented languages (Java). With the maturity and acceptance of industry-defined standards such as CORBA, Web standards such as HTTP and Java and the world-wide distribution medium of the internet, we can therefore build a open, interoperable, reliable, scalable Object-Web computing architecture for VLSI CAD.

The Object-Web architecture (Figure 3-1), is a three tier architecture consisting of the client tier, the application and data tier, and the middle tier providing the middleware services. The Object-Web architecture provides standardized multimedia document formats (HTML and MIME) and also enables the single uniform object-oriented view of distributed and heterogeneous systems integration. The combination of Java and CORBA enables quick re-targeting of existing applications on the Web.

- **The Client Tier:** The client tier is built on Web browsers to provide a standard graphical

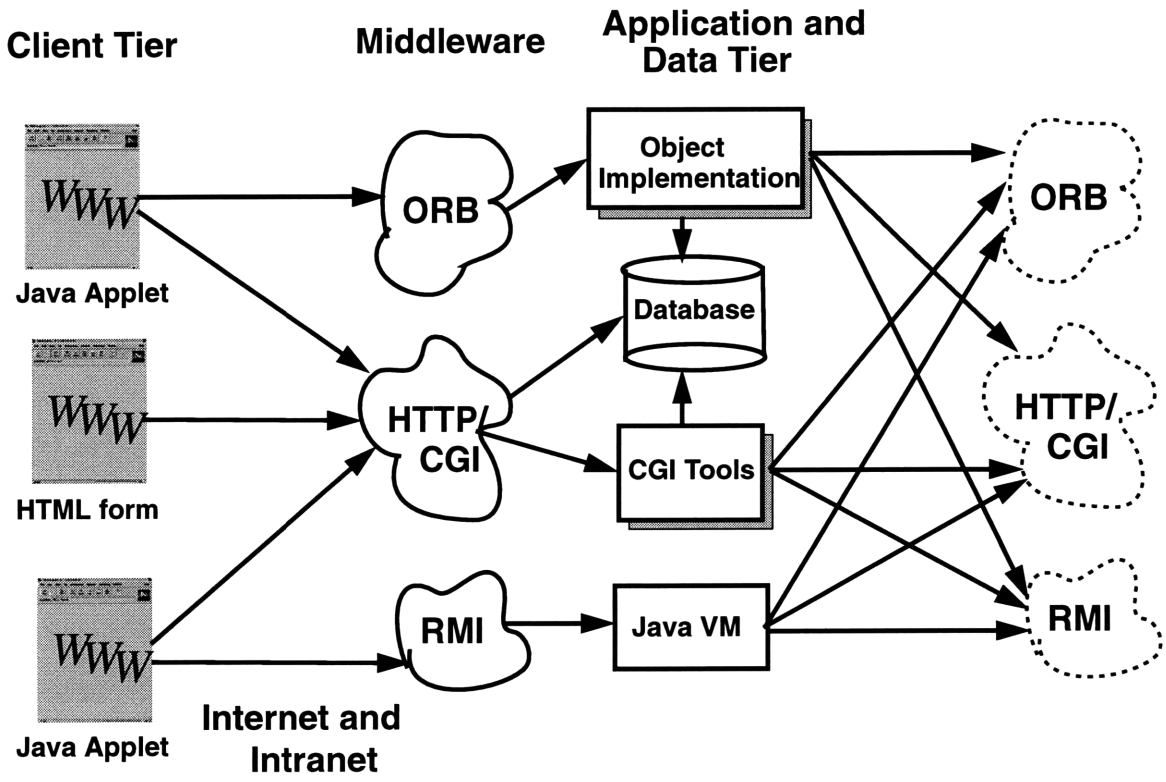

Figure 3-1: Object-Web Architecture for Distributed Computing

interface, through which users can access tools and information via the internet and the intranets. Lightweight Java client applications can run in Java-enabled browsers. Application can also be accessed using the HTML form based interface.

- **The Application and Data Tier:** The application and data tier consists of different application tools and databases distributed across the network. The tools are heterogeneous and run on diverse platforms. The application tools can be either tools which are invoked through CGI protocols or can be distributed objects providing their respective services. The distributed objects can be CORBA objects or Java objects running on Java Virtual Machines and can be located either at the client side or the server side. Applets can instantiate CORBA objects in the user's environment and act as remote objects in the application tier. Communication between the applets, which instantiate CORBA objects, will be supported by the ORBs in the middleware. The Java objects, instantiated either at the client or server side could communicate with each other through the Java RMI mechanism of the middleware services. Any service or application can make use of the other distributed services in the

application tier through appropriate middleware services.

The application tools have well defined interfaces and they cooperate to build distributed applications through the middleware services. The Java and CORBA objects have specific interface definitions to provide a global uniform interface specification, and we will describe in a later section how to define a uniform specification for CGI tools.

- **The Middleware Services:** The middleware services such as CORBA ORBs, HTTP Common Gateway Interfaces or the Java Remote Method Invocation (RMI) connect client users to resources and applications in the application tier. Client and server objects alike can send messages to other objects throughout the network using ORBs or invoke remote applications using CGI or RMI. Applets invoke methods on CORBA objects using IIOP through the ORB's, and they use Java RMI to invoke methods on Java objects running on some Java Virtual Machine. Web servers also provide access to HTML documents and Java applets, using HTTP. Applets or objects, requiring a service from a tool which is a CGI application, also use the Web server and its HTTP-CGI protocols.

  The most common form of accessing services involves a sequence of interactions between the user and the servers. In the simplest case, all interactions are via the HTTP with a Web server. The middleware in the Object-Web architecture, allows more sophisticated service delivery, involving highly interactive client-side processing (e.g., Java), as well as provides a means for the clients to communicate with the distributed object services over the internet and also gain access to application tools running on the Web server using the Common Gateway Interface.

Coupling the Web, Java and CORBA has a number of benefits. The CORBA developers and vendors gain access to the rapidly growing markets created by the Web, and the Web world gains access to services built using CORBA capabilities, which are more powerful than the simple model built on push and pull of HTML pages employed in the Web. Integrating the two worlds makes the best of the available standards rather than requiring new ones to be defined. As the Web continues to evolve, it's role as a means of delivering information is expanding to include delivery of interactive services. We list some of the advantages that the confluence of the technologies has to offer:

- Coupling the Web and the object technologies helps us to utilize tools through gateways (CGI) as well as object invocations in a transparent fashion. It also provides a way to deliver

legacy applications in a platform independent manner. The use of object technologies or the CGI depends on both how easy it is to wrap the services and also on how strongly or loosely typed are the interfaces defined by the applications.

- The convergence of the technologies, delivered on a standard user interface platform, provides the user easy, uniform and standard accesses to tools of diverse capabilities and utilities.
- By using HTML for user interface design and distributed object technology for the underlying application, designing and coding time can be reduced significantly.
- The use of distributed object technology makes it easy to develop and manage distributed client-server software efficiently through clearly defined server and client API's.

The Object-Web architecture provides a uniform model of resource interface, and downloadable front-ends, that could be generically integrated in the Web infrastructure. The distributed object-oriented approach offers good properties to encapsulate and structure Web components in the same paradigm. It brings modularity, flexibility and abstraction into the Web. Overall, the Object-Web architecture is flexible and enables tools and applications to effectively communicate with each other to build a distributed framework for Web based CAD.

3.2 Infrastructure for Hierarchical Web-based CAD

In this section we will present a formalism of applications which would help us to integrate tools in the distributed framework. Although this applies for any tools, we develop the specifications for the less strongly typed CGI based tools. CGI tools are loosely typed, because the mechanism involves data transfer (through GET and POST) only as ASCII bytes, where as methods in Java, or functions in other programming languages support other data types (e.g., integer, float etc.) to be used in parameters and variables. For tools wrapped around objects, it may be advantageous to use the distributed object technologies.

Any application can be sliced into many independent applications based on its inputs and outputs. In practical interactive tools we have the user entering some input data, getting some output results and then again entering new information and the application processes them to provide the outputs. It is very useful to visualize or approach distributed tool design with the applications sliced into independent sub-applications based on the input and outputs. Therefore any application A can be sliced into different sub-application in the form $< A_1, A_2, ..., A_n >$, where

each A_i can be visualized in the form of $< Input_i, Process_i, Output_i >$, where $Input_i$ is the set of inputs to sub-application A_i and $Output_i$ is the set of outputs of A_i. $Process_i$ is the set of steps (the algorithm and the program) used to process $Input_i$ to produce $Output_i$. The slicing is based on the interrelationship between the Output sets and the Input sets. Each $Input_j$ can be obtained only after $Output_{(j-k)}$ is calculated, for all $A_{(j-k)}$, on which A_j is dependent. In the case where $k = 1$, there is a straight line data dependence between the Outputs of one sub-application and the inputs of the next one, and we can visualize these sub-applications as functional units where there is a data dependency between each successive pairs. Also, if the $Input_i$ did not depend on the $Output_{(i-1)}$ then sub-applications i and $(i-1)$ could be combined to be a single sub-application. In other words, Input sets can be prompted to the user only after the previous applications Outputs have been processed.

It is also possible that in such a slicing, we obtain a dependency graph where an application is not dependent on some of the previous applications. In that case it may start executing as soon as all the predecessors in the dependency graph has been scheduled. This is analogous to a Control and Data Flow Graph, where the graph is determined only through data dependencies in terms of the inputs and outputs. We can have a scheduling of the applications by performing a topological sort on the applications. Each of the sub-applications can be scheduled to be executed remotely or locally depending on where the *service* is available. This form of slicing of applications is very similar to the workflow approach [14], where dependency graphs represent an executable work. In our case, more importance is given on the dependency between tools, so that we can develop a framework, where tools can be seamlessly integrated within other tools.

The following example shown in Figure 3-2, expands on this formalism. Let us consider the following CAD tool, which allows the user to draw the schematic of the circuit, then extracts the schematic and produces a spice netlist and then simulates the circuit and produces an output plot of the simulation result. Such an application can be thought as a collection of applications like < *Schematic Editor, Spice Extractor, Simulator, Graph Utility* >. The Schematic Editor uses GUI to enter a schematic and produces a representation of the schematic. The Spice Extractor takes the output of the Schematic Editor and produces a spice netlist. The Simulator takes input the Spice netlist and a set of input vectors and produces a data set of the output. The Graph Utility takes in a data set and the output format (Postscript, Jpeg or Giff) and produces an image file. Our main application acts as an agent to properly invoke each tool with the output of the previous tool.

The input and output sets are sets with elements of the form,

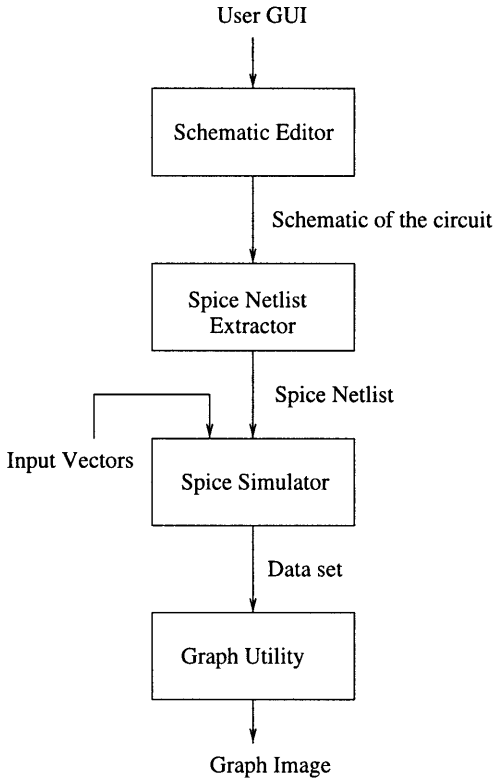

Figure 3-2: Slicing of Applications based on Inputs and Outputs

```
<Parameter identifier>  <Parameter value>
<File identifier>       <File data>
```

In loosely typed systems, any application's input and output can be specified with the help of these two elements. Each Sub-application A_i is independent in the sense no other information other than its Input set is required to execute the application. The division is also more of a functional division, where each sub-application performs a specific function according to the algorithm of $Process_i$.

Such a standard description of applications helps us to define an application very specifically in a distributed environment. Although such a slicing of interactive tools may be intuitively obvious but such a formalism helps us design a distributed framework due to the following reasons:

- Each of the independent sub-applications could be developed separately in a distributed manner.
- The clear specification of the applications in terms of inputs and outputs provides a first step to easily integrate them in a distributed framework.

- Many applications need common sub-applications. Duplication of effort in terms of development and management can be reduced if the capability of such application tools are utilized. In a later section we would build on this concept to describe a hierarchical Web based CAD framework.

We can easily see in a distributed environment, such an application A could be a design agent and the sub-applications are distributed point tools. The design agent interacts with the designer, invoking tools and accessing data on the designer's behalf [2]. Such a design agent could be a static one in which the agent is responsible for interpretation of outputs and appropriately generating the inputs for the chain of sub-applications. Or the design agent could be a dynamic one which dynamically configures itself (in terms of the sub applications that constitute itself).

3.3 HTML Forms and multipart/form-data

In the previous section we mentioned that the Input and Output sets could be sufficiently specified in terms of a *tag* (Parameter or File) and the *value* (the value of the parameter or the file contents). In this section we describe the Web technologies currently available to support this format. As we mentioned previously, this applies specifically for tools invoked as CGI programs.

HTML forms are the primary method (and also the oldest) for making a web document interactive. Forms allow the input of information by a reader. Various input methods exist to enter this information from the user such as buttons, pull-down menus and text boxes. When a user has finished entering information in an HTML form, they can "submit" this information, which results in execution of a script at the server as specified in the form.

The most important consideration of a form is how it should send the information input to the server. We described briefly in Chapter 2 that there are two methods available to send information to the server, GET and POST method, which is specified by the `METHOD` tag of HTML forms. With `METHOD="GET"` most documents are retrieved from the server by requesting a single URL. With the GET method all information needed is appended onto the end of the URL. `METHOD="POST"` is usually more useful when the client wants to send more information (e.g., upload a file). The `ACTION` tag of the forms tells the browser where to send the information. The `ACTION` tag always point to a CGI script. If we want to send our information input to a CGI program called feedback.pl in the cgi-bin path we will enter the following line,

```
ACTION="/cgi-bin/feedback.pl"
```

The traditional forms encode the user input data with encryption "application/x-www-form-urlencoded" type. A more versatile and useful encoding type "multipart/form-data" has been recently introduced and this can handle arbitrary file uploads. This is indicated by the tag ENCTYPE, which if unspecified, defaults to "application/x-www-form-urlencoded". The HTML tag for such a form could be:

```
<FORM ENCTYPE = "multipart/form-data" ACTION="_URL_" METHOD=POST>

<INPUT NAME="userfile" TYPE="file">
....
```

This additional capability of file-upload for the browsers is very useful and necessary for Web based CAD applications where huge binary files need to be submitted and retrieved. We propose to use the POST method and ENCTYPE multipart/form-data as standards for Web based CAD applications. The following reasons could be attributed to support our proposal:

- The POST method and ENCTYPE multipart/form-data is capable of handling any arbitrary data transfer between clients and servers.
- After studying a large number of input and output specification of various tools we found out that it is sufficient to specify the inputs and outputs of tools with only two tags. These two are parameters and files. Parameters could be thought as a condensed version of files for the ease of understanding. In fact every field could be handled as a separate file, but the distinction is useful for practical purposes. Therefore any tool could have all its inputs specified in the form

  ```
  <Parameter identifier1>  <Parameter value1>
  <Parameter identifier2>  <Parameter value2>
  ........................
  <File identifier1>       <File data1>
  <File identifier2>       <File data2>
  ```

  The semantics of multipart/form-data is best suited to capture both of these tags. It is useful to keep these tags as simple as possible because that would facilitate communications between different heterogeneous tools.

- The media-type multipart/form-data follows the rules of all multipart MIME (Multipurpose Internet Mail Extensions) data streams as outlined in RFC 1521 [21]. This makes the rules of communication very standard and powerful. multipart/form-data contains a series of parts. Each part is expected to contain a content-disposition header where the value is "form-data" and a name attribute specifies the field name within the form. Each part could potentially handle different mime type and each part could be encoded separately with "content-transfer-encoding". File inputs may also identify the file name using the 'filename' parameter of the "content-disposition" header.

  For the discussions we present a small example of how an HTML form with two input tags would encode the data using the media-type multipart/form-data.

```
<HTML>
<FORM action  = "/cgi-bin/test/post.pl"
      ENCTYPE = "multipart/form-data"
      method  = "POST">
<INPUT type="text" name="Name" size=30>
<INPUT type="file" name="example_file">
<INPUT type="submit" Value="Submit">
</FORM>
</HTML>
```

  The following is the data sent to the server after we submit the form.

```
POST /cgi-bin/test/post.pl HTTP/1.0
Content-type: multipart/form-data, boundary=boundaryString
Content-length: 217

--boundaryString
Content-Disposition: form-data; name="Name"

Debashis Saha
--boundaryString
Content-Disposition: form-data; name="example_file"; filename="file1"
```

```
filedata comes here
--boundaryString--
```

We note that the line separator is <CRLF> which is the carriage return character followed be the linefeed character ("\r\n" in Unix). We also note the header the client sends to the HTTP server as the protocol of communication between clients and servers.

- The ENCTYPE multipart/form-data is an easy and versatile way of sending arbitrary data. It is essential to have compatibility of input data to CGI programs from HTML forms and other clients (design agents, Java clients etc.). For example, if a designer wants to call a CGI form based tool from his internal applications, standardization of the encoding type would allow him to exchange data in a transparent way. Moreover this could serve as a common exchange protocol for inputs and outputs of applications.

3.4 The Application Tier - Web based Point Tools

The application tier of our Object Web architecture consists of distributed tools invoked either as remote objects or as CGI programs. We first present the characteristics or capabilities of any back end tool in the framework and then we will go onto how to use these features to build an efficient hierarchical CAD framework over the Web.

Any point tool or application on the Web can be specified as, $< Input_{tool}, Output_{tool} >$, where $Input_{tool}$ is the input set and $Output_{tool}$ is the output set expressed in the standard form as described in the previous section. We call this set **Basic Interface Specification (BIS)** of the tool. The $Input_{tool}$ can be obtained by a CGI form or a Java based GUI. The input is then given to the tool to be processed and produce the $Output_{tool}$. The tool itself can be a CGI program, linked to a HTTP address or the same Java applet or a different Java applet, which processes the input. Having the BIS independent of the implementation of the tools, helps us to build a platform independent framework. Also, in VLSI CAD applications, a strongly typed interface specification as in IDL of the CORBA is not always necessary. Therefore, for many CAD tools, which are invoked as CGI programs, such a *down casted loosely typed IDL*, is sufficient. Moreover, the basic Web infrastructure (HTTP and CGI) is not designed to support a strongly type application interface. And as we mentioned before, the specifications of multipart/form-data are strong and very efficient

to capture the BIS of a CGI tool. For example for the *Graph Utility* application of the example CAD tool in Section 3.2, the BIS could be:

```
Input:
    <Parameter OutputImageType{Jpeg, Postscript, Giff} >
    <File InputDataSet>
Output:
    <File OutputImage>
```

We can have a parameter taking one of several specified values, which are specified in the BIS. For example the parameter OutputImageType can take any of the three values specified. Any other value would be considered invalid. The tool takes an input of the type `File`, identified by `InputDataSet`, a `Parameter`, which specifies the output format of the image and the `OutputImage` is the identifier of the output file.

3.5 CGI back-end Tools

Let us consider that our back-end processing is done by a CGI program which is bound to a HTTP address like, "http://apsara.mit.edu/cgi-bin/pythia/pythia.pl". The input to this CGI program can be provided by a Java applet or a HTML form. Both the input agents or processes provide the data in the multipart/form-data and they use the POST method to talk to the HTTP server, which would execute the CGI program. Any tool (back-end application) should be capable of decoding this multipart/form-data, process the data, and produce the output. In addition to the BIS we add another mandatory parameter to the input set of the tool :

```
<Parameter OutputType, Value HtmlType/BasicOutputType/URLOutputType>
```

The parameter `OutputType` currently supports three values: `HtmlType`, `BasicOutputType` and the `URLOutputType`. If the tool was used as an end tool, (i.e., the display is intended for the user), it may have the **OutputType** parameter set as **HtmlType**, in this case, the tool produces a dynamic customized HTML page of the output. This is the case of a stand-alone tool which prompts the user with a HTML form and produces a HTML output for the browser.

In addition to HtmlType, every tool in our framework should be able to produce the output in multipart/form-data, when the **OutputType** parameter is set as **BasicOutputType**. This is necessary because we would like to build a hierarchy of tools using this existing tool. At least

there should be the potential to integrate this tool with other tools on the Web. The standards of multipart/form-data provides a global and standard encapsulation of tool output.

The OutputType of **URLOutputType** produces a URL which contains the output of the tool. This is necessary because applets may not have the capability of parsing a HTML file, but it can prompt the browser to load any URL which parses the HTML file. Moreover, the URLOutputType saves some bandwidth if the next tool in the pipeline needs the same data, since it is efficient to send the URL of the data, rather than the whole data set. This is specially significant for VLSI CAD applications, where designs moving back and forth in the network are quite big (some hundreds of Kilobytes of data).

Therefore the HTTP address bound tools have the following capabilities:

- All the input data (the data for its input set) is sent in the form of multipart/form-data and the tool needs to decode this data to its internal representation of data.

- Every tool has a parameter OutputType in its BIS which can have three values (at present) as HtmlType or BasicOutputType or the URLOutputType.

- It has the capability to produce a custom HTML output page or the output set of the BIS as multipart/form-data according to the OutputType parameter in its BIS.

- In its BasicOutput, the tool could have any headers which the HTTP server generates. In addition to those it also generates a header containing the information of the output generated, which at present contains just three tags, BasicOutputType (which specifies the address of the program generating the output) and Content-length (which specifies the content length of the multipart data in bytes) and Content-type (which indicates that the data type is multipart/form-data). Therefore a typical output generated by a CGI tool in our framework would look like:

```
BasicOutputType: <address of the application>
Content-type: multipart/form-data, boundary=<boundaryString>
Content-length: <Content length of the multipart/form-data>
<CRLF>
<CRLF>
contents of the multipart/form-data
```

3.6 Hierarchical Web Tools

In this section we show how the point tools described in the previous sections could be used to build a hierarchical Web based CAD framework. Given the capabilities of all the HTTP address bound tools, we can build tools which would utilize the capabilities of the existing tools and serve as a new tool on the framework. By HTTP address bound tools we mean that if a socket connection is made to the web server and the input data as specified in the BIS of the tool is POSTed as multipart/form-data to that address, the HTTP server executes the tool and writes back the output to the socket as specified in the BIS.

Let us have a tool X on the Web which is bound to the HTTP address **http://x**. And let us have another tool Y on the Web on the address **http://y**. Given these two tools and their BIS, we can easily construct another tool Z bound to **http://z**, which uses tools X and Y to produce its own output. To the external world Z acts like an independent tool which has its own BIS. Therefore any client can make a connection to http://z and get the desired output of the tool, without being aware of the fact that Z actually calls two distributed tools on the Web. The series of action are described in details below and is shown in Figure 3-3.

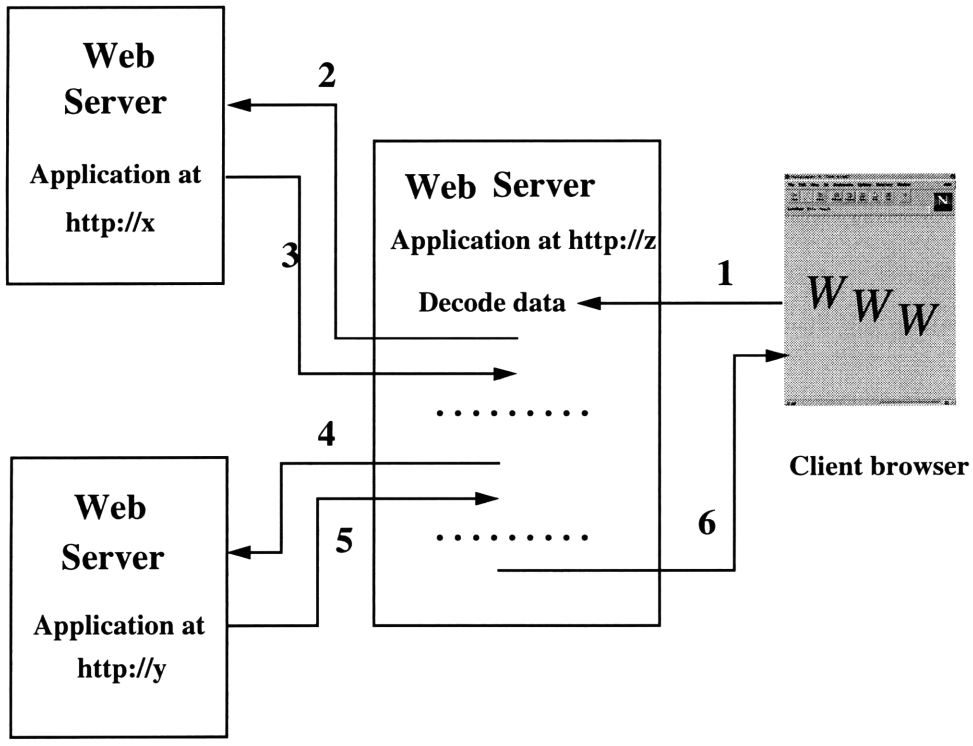

Figure 3-3: Data flow in Hierarchical tools

1. We have a CGI program Z bound to http://z. Now Z first gets its own input from an applet client or a HTML form.

2. It decodes the input which is in form of multipart-form data and is POSTed to the Web server. It processes the input and makes a socket connection to the Web server at http://x. It packs all the input data specified in the BIS of X and writes it to the socket and waits for the output.

3. The server at http://x executes X. X produces the Output as specified by the output set, and the server writes back the data to the socket.

4. Z then decodes X's output, processes the data and then makes a similar call to http://y, sends the inputs as specified in the BIS of Y and waits for the reply.

5. The server at http://y executes Y. Y produces the Output as specified by the output set, and the server writes back the data to the socket.

6. Z produces its own output as specified by its BIS and sends it to the client browser.

It can be seen the only thing that needs to be known is the BIS of the individual tools. All other protocols are well defined, and therefore communication in this distributed framework can be done in a uniform and unambiguous way. Now Z could be called from another CGI tool, therefore we can build a hierarchy of tools on top of the existing ones. As we speculate an enormous increase in the Web access to different CAD tools in the coming years such a distributed framework could help utilizing different existing tools on the Web. The framework provides the protocols and standards needed to effectively do distributed design on the Web.

3.7 Java and Web Tools

Java applets could be used in the client tier, where the applets are embedded in the Web browsers instead of HTML forms to enter the input data of a tool. The applets also serve as an independent tool, doing some computation or service at the client side. Java objects can also be located at the server side and provide some service to applets in the client tier or other tools, through remote method invocation on them.

The current day implementations of the Java Virtual Machines has many security restrictions, which limit the applets accessing client resources. Although the JDK1.1 release has the digitally

signed applet security model, the mechanism is in its early stages of maturity and is not widely used, nor do many browsers support them. It will be helpful to understand the limitations of this security model.

The security restrictions do not allow browser applets to do a file read and therefore file uploads are not possible with only browser applets. To upload files to a Web tool, we need to have a mix of HTML forms and Java applets. We could possibly have a helper tool linked with the web server which first uploads a file with a HTML form and then sends a dynamic Web page with an applet to enter the remaining inputs for the tools. The applet can then talk to any tools available on the Web with the format specified and get the results back. Therefore Java applets could act as agents which would help use the capabilities of different Web tools.

The present day security restrictions also do not allow applets to open a socket with any machine other than the base machine (from which the classes were downloaded). Therefore to efficiently utilize any tools anywhere on the Web it is necessary to have helper tools linked with the server with which the applet communicates. In the Object-Web architecture we have the client tools communicating with any tool in the application tier with the help of the middleware. But the above security restrictions call for a proxy mechanism to go around the problem. For example, if an applet is loaded from http://x and it wants to use a tool available at http://y, we need to have a proxy server at the machine X, which takes the inputs from the applet, invokes the call at http://y and returns the result. The proxy mechanism is shown in Figure 3-4. In the figure, the dotted line shows the intended communication, and the bold lines show the tool communication required due to the security mechanisms. Although such proxies do not hinder distributed computation fundamentally, but they add the latency and decreases the transparency to some extent. But with the new Java security models emerging, we are optimistic that the limitations will be solved without sacrificing security.

We use the same encryption of multipart/form-data for the applets to communicate with tools and vice versa, which provides a standard packing of data for any Web based applications. The applets POST any data to the server tool as multipart/form-data and the server tools write back the results and data to the applets again as multipart/form-data or as a simple URL, which the applet displays in the browser. We present a prototype of such communications in the Section 3.8.

We can also build Java applet tools which are independent tools. These Java applets can either provide some stand alone services or make use of some remote services and tools. In VLSI domain, we will like to have Java tools, which handle the complex user-interfaces, but which are not

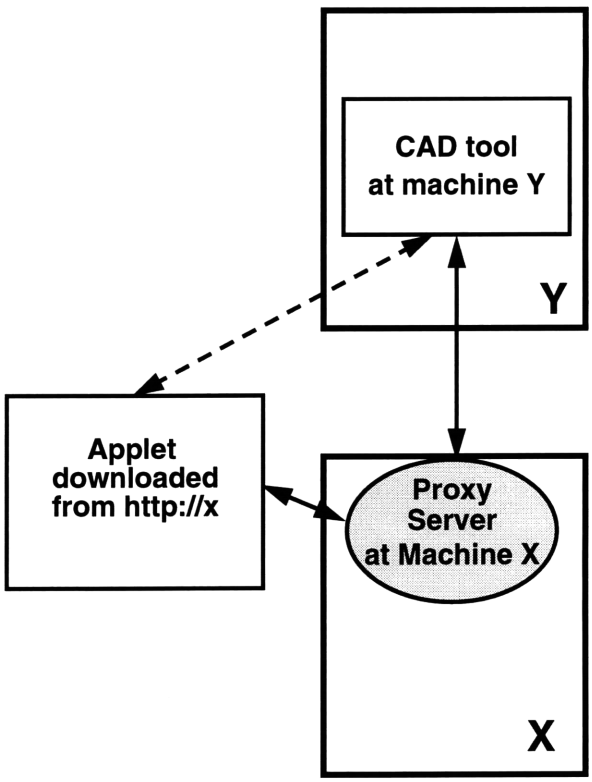

Figure 3-4: The Proxy Mechanism

computationally intensive. The Java applets should utilize the remote services of the application tier, which usually would run on more powerful machines.

3.8 Example of Web based Hierarchical Tools

In this example, we demonstrate the frameworks hierarchical capabilities. Consider two point tools, Pythia [29] and a graph plotting utility.

1. **Pythia:** *http://apsara.mit.edu/cgi-bin/pythia/new-pythia.pl.* This is a power estimation tool, which takes a verilog netlist and other parameters (Technology type, Cell models) and generates energy and power information [29]. It also generates a data set of the current drawn and the power consumed at different time intervals. The BIS Output specification of Pythia contains *current_data_file*, and *power_data_file*, which are files containing the data sets of the current drawn versus time and power dissipated versus time, as a list of X and Y co-ordinates. Listed below is the complete BIS of Pythia.

```
// BIS of Pythia
// Location: http://apsara.mit.edu/cgi-bin/pythia/new-pythia.pl

Input:
    <File netlist>
    <Parameter tech_file>
    <Parameter cell_file>
    <Parameter current{on/off}>
    <Parameter power{on/off}>
    <Parameter glitch{on/off}>
    <Parameter glitchwidth>
    <Parameter OutputType{HtmlType/BasicOutputType/URLOutputType}>
Output:
    <Parameter power>
    <Parameter energy>
    <File current_data_file>
    <File power_data_file>
```

Pythia can be accessed from the HTML form at, *http://apsara.mit.edu/pythia-doc/new-pythia.html.* Figure 3-5 shows a snap shot of the Web browser with the results of running Pythia on a particular Verilog netlist.

2. **Graph Plotting Utility:** *http:// apsara.mit.edu/ cgi-bin/graph/ drawgraph.pl.* This tool takes in a file of X,Y data sets and produces a image of the plotted graph as a Jpeg image. In the BIS of the tool we have *datafile* which is a file containing the data set and *graphfile* is the Jpeg image file. The parameter OutputImageType tells the script the format of the output image. The BIS of the tool is listed below:

```
// BIS of Graph Plotting Utility
// Location: http://apsara.mit.edu/cgi-bin/graph/drawgraph.pl

Input:
```

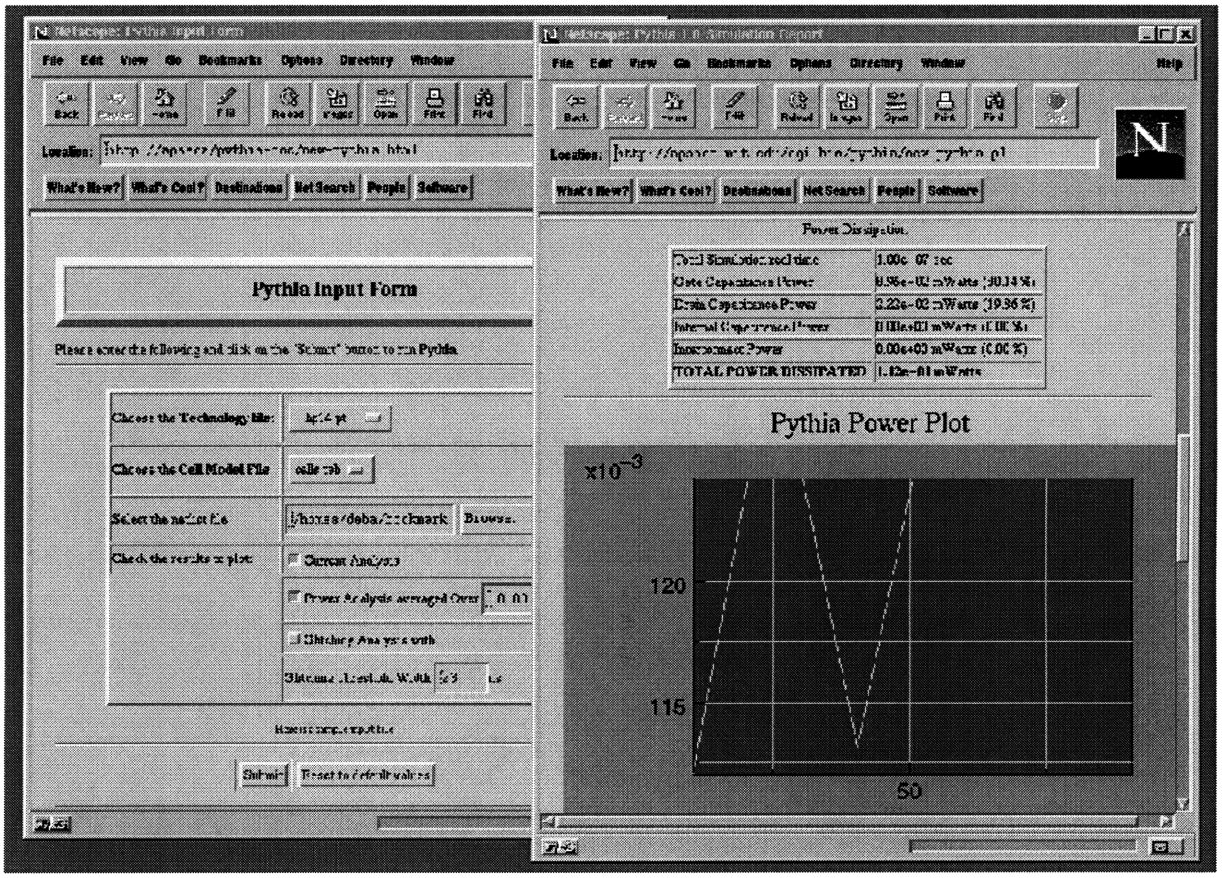

Figure 3-5: Snap shot of Pythia

```
    <File datafile>
    <Parameter OutputType{HtmlType/BasicOutputType/URLOutputType}>
Output:
    <File graphfile>
```

The tool can be accessed at: *http://apsara.mit.edu/graph-images/drawgraph.html*

Now that we have these tools on the Web we make a new tool, which takes in a verilog netlist and plots the current drawn by the circuit. It produces a plot of the current versus time. This tool again is an independent tool with its own BIS, but internally it calls Pythia to get the data set of the current drawn by the circuit and then calls the Graph Utility to plot the data set. This new hierarchical tool can be accessed at: *http://apsara.mit.edu/demo1.html* and is bound to the URL *http://apsara.mit.edu/cgi-bin/demo1/demo1v2.pl*. The flow of information is similar to that depicted in Figure 3-3 where X is Pythia, Y is Graph Utility and Z is the hierarchical web tool.

Although the example is a trivial one, it demonstrates, how to encapsulate different CGI tools from within other tools. We can expect to see a lot more of CAD tools being available on the Web as CGI scripts, therefore such a hierarchical framework will allow us to integrate any tool, anywhere in the world in a transparent fashion.

Pythia, also demonstrates, how a CGI tool can utilize an existing applets. Pythia itself also uses a graph plotting applet to draw the current and power data sets. We modified a freely distributed Java2D Graph package [10], which is a package of Java classes designed to facilitate plotting data using Java applets. The data can be calculated by the applet or be loaded from a file given a URL. When the `OutputType` is specified as `HtmlOutput`, Pythia generates a dynamic HTML page, with an embedded applet, the parameters of which contain a dynamically generated URL, which contains the data set. Thus, applets can be embedded in the results of CGI programs, with the data passed as URLs.

Chapter 4

WebTop: Distributed Microsystem Design Center

Future microsystem design environments must integrate globally distributed cell libraries, models and design tools

In this chapter, we describe WebTop [26], a Java hierarchical schematic/block editor, which is the entry point of the distributed microsystem design framework. WebTop supports hierarchical cells and multiple views of a cell. WebTop allows cells to have Verilog or SPICE views in addition to the schematic views. WebTop has hierarchical netlisting capabilities for both Verilog and SPICE. WebTop also supports distributed cell access. Cells could be stored in different Web servers and loaded as URLs. WebTop has interfaces to other Web tools and demonstrates how an Java applet tool could utilize other CGI based tools and distributed objects on the Web.

WebTop is developed in Java. WebTop is based on a public domain applet called "Digital Simulator" (DigSim) [6]. The first version of WebTop and its implementation and User's Manual are described in [16]. WebTop has been modified with many more advanced features and has been supplemented with a framework for distributed library and tool access. WebTop along with the Web based framework, distributed cells and remote tools constitute the Distributed Microsystem Design Center.

4.1 Overview of the Distributed Design Center

In this section, we present an overview of the features of the Distributed Design Center, which consists of WebTop, the editor including the library manager, the distributed cell access mechanism, the netlister, distributed Web tools and WebTop's interfaces to distributed tools.

The Editor

WebTop includes a full hierarchical schematic editor, with a simple graphical user interface. It includes mechanisms to add and delete cells, cut and paste schematics, change properties of cells and mechanisms to save cells. It includes a Color Manager, using which the user can customize the colors of different components. WebTop also has a number of primitive cells (logic gates, MOS transistors, resistors, power supply, etc.) with which a user can build other cells. WebTop also supports block editing, in which the user creates a template of the cell (in terms of the input and output pins), and provides a behavioral or structural view in SPICE or Verilog. The new cell can then be included within other cells.

The Library Manager

The Library Manager keeps track of the different cells and their views. A cell can go down to any level in the hierarchy. In other words, there is no limit to the depth in the hierarchy. Cells can have both Verilog and SPICE views in addition to their symbol and schematic view. Currently, Webtop does not include a symbol editor. The symbol for the cell is automatically generated by the editor.

Both the Verilog and SPICE views can be in lined, or referred to by a URL or a location of a local file. For example, the Verilog view of a cell may be `http:// apsara.mit.edu/ VERILOG/ input-control.v`, which tells the library manager that the view resides in that URL. If the view was specified as `file:/homes/deba/CellStore/input-control.v`, the library manager will read the local file when the view is to be displayed. Figure 4-1 shows a snap-shot of the Library Manager with the multiple views of the cells.

Although WebTop currently supports only Verilog and SPICE views, it can be easily modified to include other views (for e.g., VHDL). Currently, this would require changing the code to support the additional views. We will also need to write a new netlister or extractor for the additional views. WebTop can also be easily modified to dynamically add new views and interface with distributed netlisters for the added views.

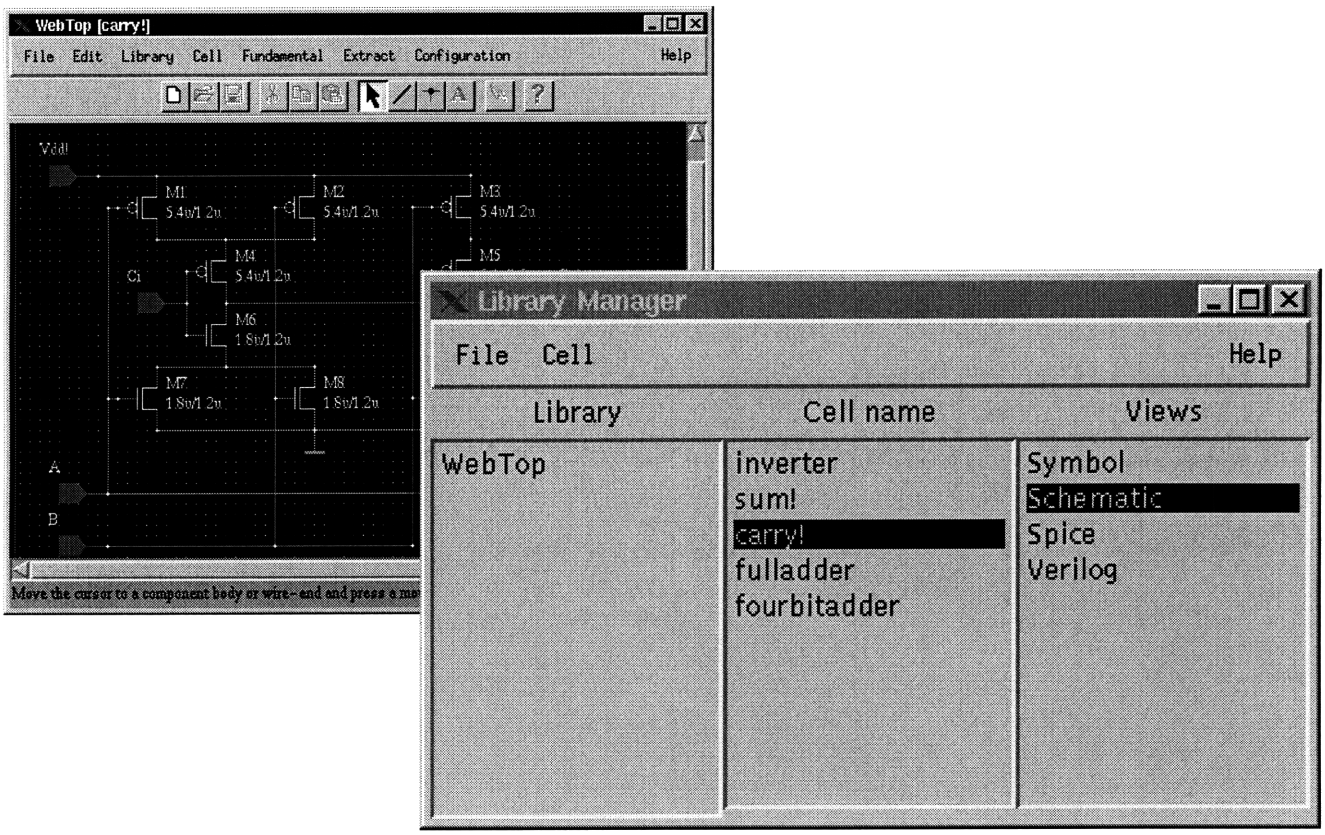

Figure 4-1: WebTop: Library Manager

Distributed Cell Access

WebTop supports distributed cell storage and access. Cells could be stored in different Web servers. Cells in WebTop can be loaded either locally from a file or can be read from a URL. This directly conforms to a distributed microsystem design framework where cells could be stored and accessed over the internet.

A CellServer has been implemented which allows users to store cells remotely at the server's site using login and password verification. Users have the ability to access, modify and delete cells in their respective directories. All cells saved at the server's site are automatically linked as URLs and accessible for use for all other users. Only the owner of the cells has the permissions to modify or delete the cells. Figure 4-2 shows a screen dump of the editor, the library manager with the URL cell load dialog box.

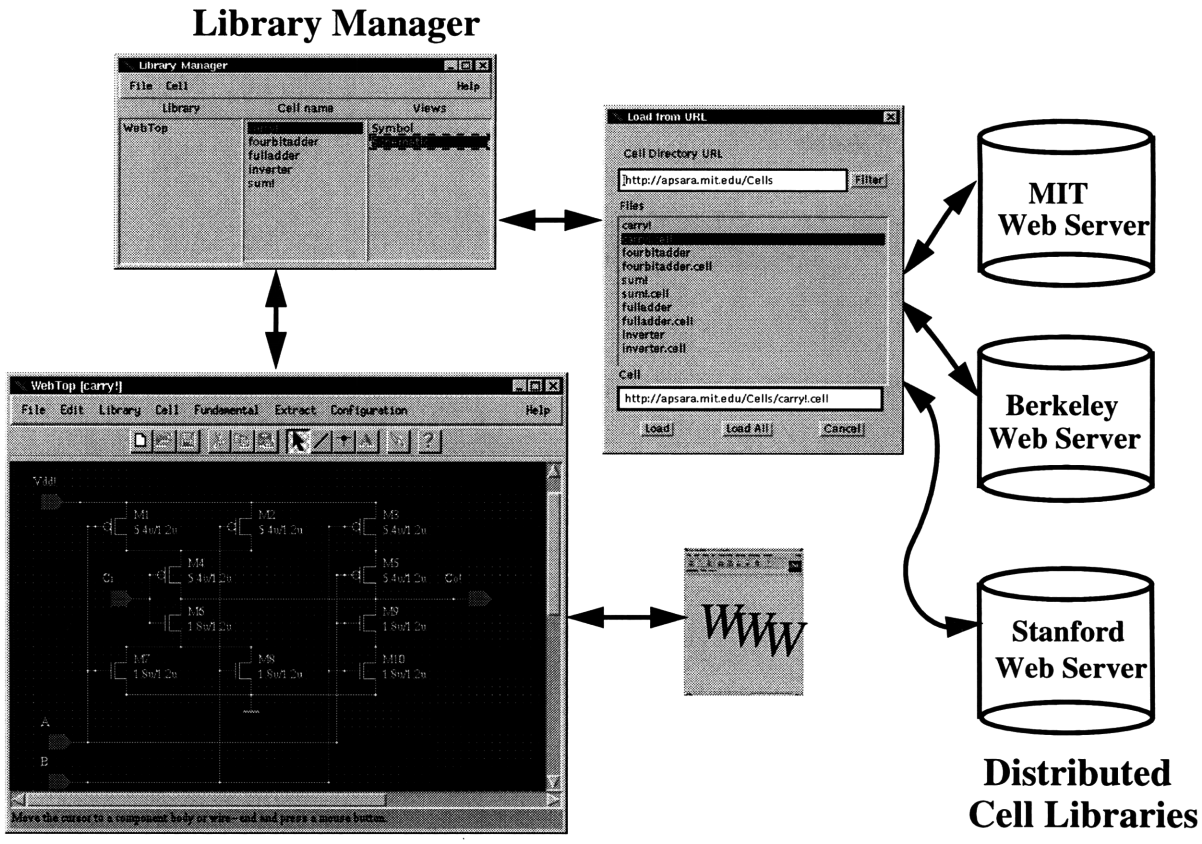

Figure 4-2: WebTop: Distributed Cell access

Netlister

WebTop has a hierarchical netlister for Verilog, SPICE and PowerPlay [15, 19]. The netlister for PowerPlay is in the experimental stage. The netlister goes down the hierarchy in the database of the library manager. If a cell has a Verilog or SPICE view, the view is considered as a leaf view, and the netlister stops going down the hierarchy for that cell. The netlister requires that all the referred cells to be present in the library manager's database, to expand them. Figure 4-3 shows a snap shot of the netlister's output in a window. The netlister can also directly netlist to a file or save the netlist in the remote CellServer.

Distributed Tool Integration

WebTop uses the object-web architecture, described in Section 3.1, for distributed tool integration. CAD tools in the application tier can be either CGI programs, Java objects or tools wrapped around as CORBA objects. The standard specification of RMI and CORBA makes it very easy to

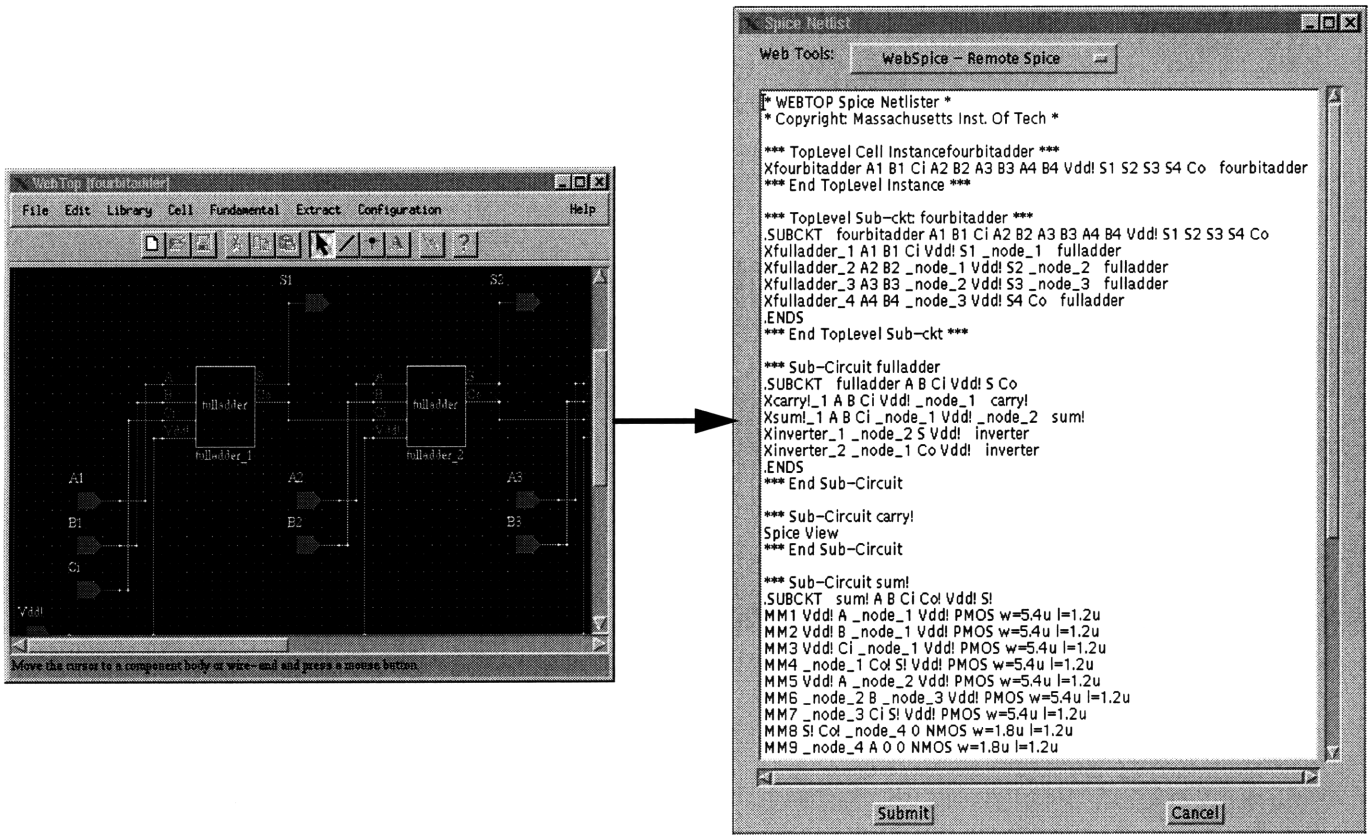

Figure 4-3: WebTop: Netlister

make transparent calls to tools which communicate using RMI or CORBA. Tools can also be CGI programs and could be integrated using the framework described in Section 3.2.

WebTop has interfaces to different Web tools and demonstrates how an Java applet tool could utilize another CGI based tool on the Web. WebTop interfaces to a CGI based point tool **WebSpice** at `http://apsara.mit.edu/cgi-bin/spice/spice.pl`, which provides Web access to the circuit simulator HSPICE. WebTop also interfaces to Pythia, a verilog power estimation tool [20] and PowerPlay [19], a system level power estimation tool. WebTop also makes use of the RMI mechanism to interface with a *generator tool* for some modules of a power supply. The inter-tool communication is generic and can be used for any tool accessible over the internet. Each server side tool could itself call other Web tools in our hierarchical framework. Figure 4-4 shows the tool integration architecture of the distributed framework. In Section 4.2, we will describe in details the tool integration mechanisms of WebTop.

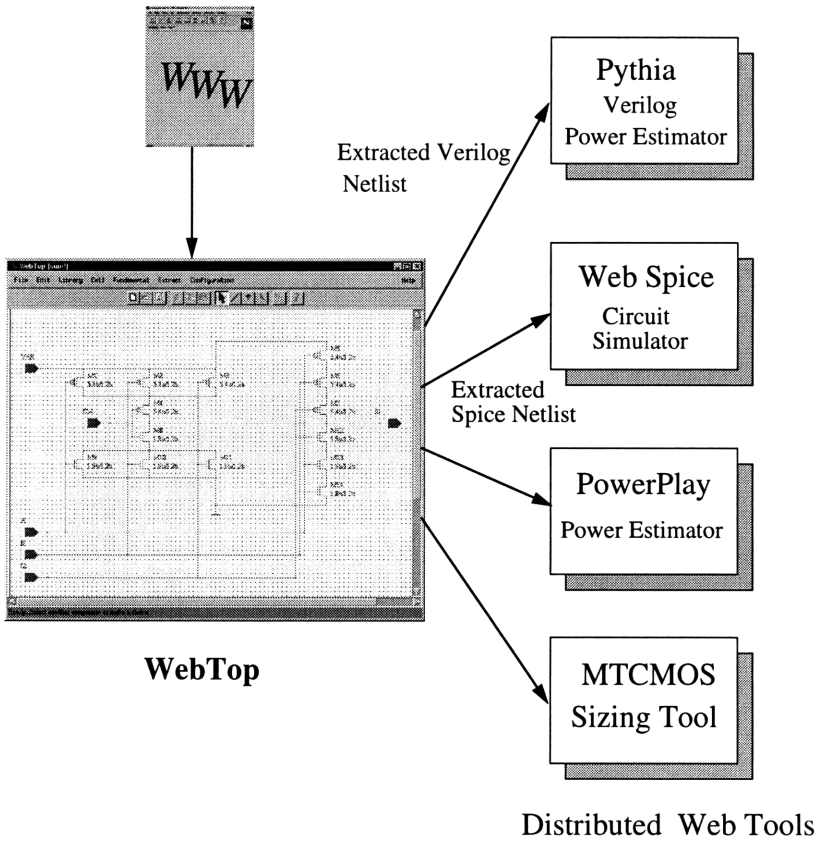

Figure 4-4: WebTop: Distributed Tool Integration

4.2 WebTop: Implementation and New Features

In this section, we will describe some of the new features of WebTop and the relevant implementation issues. There has been major enhancements to the original WebTop described in [16]. The editor has been extended to handle buses, vectored instantiation, distributed cell views and has been integrated to many Web based tools.

4.2.1 Vectored Pins and Bus

The first version of WebTop could only define input/output pins of single bitwidth. Also all the wire connections were assumed to be of single bitwidth. Presently, the editor can handle pins and wires of any bitwidth. Following are the details of usage of the pins and buses.

- The name of a pin or a wire specifies its width. Pins or wires of non-unit bitwidth are specified with $name[start\_bit : end\_bit]$, where *name* is the name of the pin or wire, *start_bit*

and *end_bit* are the non-zero integers of the starting and ending length of the element.

- There is no restriction on the *start_bit* to be the larger of the two numbers. Therefore we can have a wire $wire[7:0]$ as well as $wire[0:7]$. In the first case the least significant bit of the wire bus is $wire[0]$, whereas in the later case it is $wire[7]$.

- Any junctions, which are not connected by a wire, or connected by an un-named wire are considered to be of one bit width.

- Pins of non-unit bit width could be connected only with a similar bit-width or an unit bit-width wire. In the case where a vectored pin is connected with an unit width wire or pin, all the pins are shorted with the later. In case of a similar width wire, there is a one-to-one correspondence of the wire from the most significant bit position.

- For SPICE netlists, the buses and vectored pins are automatically expanded with an *_number* appended to the name of the pin or bus. For example, $pin[3:0]$ will be expanded as pin_3 pin_2 pin_1 pin_0.

- WebTop does not have explicit *taps*, but taps can be adequately handled in the schematics by properly naming the wires with proper vector widths and notation.

- Multiple buses can be coalesced to a bigger bus using "," (comma) in the name of the wire. For example, a wire $a[3:0], b[1:0], c, d$, is a properly named wire consisting of two buses $a[3:0]$, $b[1:0]$ and two single wires c and d. The total width of the new bus is the sum of the width of the individual buses and the order is maintained.

Appendix B provides some examples demonstrating how the Verilog netlister will netlist the schematics for different pin and bus connections.

4.2.2 Vectored Instantiation

Non-primitive (non-fundamental) cells can be vectored instantiated. This means that a cell with similar vector naming conventions as the vectored pins, will be expanded when netlisted. The pins of a vectored instantiated cell behave a little differently from the normal pins in three different ways.

1. If a single bit width wire, an unnamed wire, or a junction is connected to one of the pins, all the pins of each of the cells of the vector are shorted to the same wire.

2. If a vectored pin of a vectored instantiated cell is connected to a bus of the same width as that of the pin, all the pins of all the instances get connected to the same bus.

3. A full bus can be specified to be connected to a pin. In this case the bus is properly expanded to make connections to each pin of each cell. We note that the buses that are connected to pins of a vectored instantiated cell must be either fully specified, or be of equal width as of the pin it is connected to. Any thing else is flagged as an error.

The following example schematic of Figure 4-5, shows how vectored instances are expand in each of the cases. The cell *test* in this example has one single bit input pin ($in1$), a vectored input pin ($in2[2:0]$), and two vectored output pins ($out1[1:0]$ and $out2[3:0]$). In the top level vectored instantiated *test* ($I0[3:0]$), both $in1$ and $out1$ are connected to a single wire (a and c respectively). $in2$ is connected to a full bus ($b[11:0]$), and $out2$ is connected to a bus of the same width ($d[3:0]$). The verilog model of the cell *test* is:

```
module test( in1,in2,out1,out2 );
    input  in1;
    input[2:0]  in2;
    output[1:0]  out1;
    output[3:0]  out2;

    . . . . . .
endmodule
```

The top level cell of the Figure 4-5 is netlisted as:

```
module my_cell(  );
    wire[3:0]  d;
    wire  c;
    wire[11:0]  b;
    wire  a;

    // Sub Cells start here
    //Begin Vectored Instantiation: I0
        test I0_3( a,b[11],b[10],b[9],c,c,d[3:0] );
        test I0_2( a,b[8],b[7],b[6],c,c,d[3:0] );
        test I0_1( a,b[5],b[4],b[3],c,c,d[3:0] );
        test I0_0( a,b[2],b[1],b[0],c,c,d[3:0] );
    //Vectored Instantiation End
endmodule
```

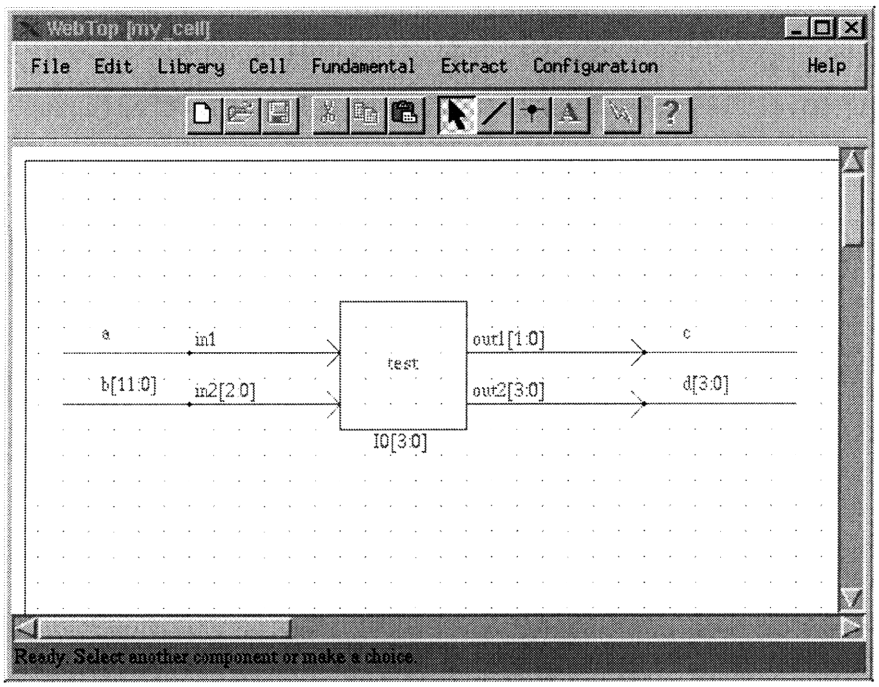

Figure 4-5: Netlisting Vectored Instances

4.2.3 Strength of Net Types in Verilog

Wires of different strengths are supported by the Verilog nets `tri0, tri1, supply0, supply1`. In WebTop's representation of wires, the strengths of the nets can be expressed in braces (enclosed within "{" and "}"). For example if we have a wire of the name {`supply0`}`vdd` all the pins connected to the net will get the name of `vdd` and the wire will be declared as:

```
supply0 vdd;
```

This convention is generic, and the netlister does not check for the validity of the net type. Therefore different net types can be supported by the wire naming convention.

4.2.4 Behavioral Views as URL's and local files

WebTop supports multiple views of a cell. The views (Verilog or SPICE), if present, act as the leaf cell views for the respective netlisters. The views can be provided in three different ways:

1. **Inlined View:** The view of the cell is provided in full text. The view is internally stored as a string.

2. **Local File Views:** The view of a cell could point to a file in the disk. This could be achieved by providing the fully specified path name of the file preceded by a `file:`. For example, a cell might have a verilog view of `file:/homes/deba/CellStore/input-control.v`, which implies that the view should be loaded from the file provided in the view.

3. **URL Views:** If the view starts with an `http://`, the view is represented by the full URL. The netlister, would load the view from the URL.

   *Due to a bug with the JDK1.02 when reading from a input stream of a URL, we require that the first line of the view in the URL should start with a comment identifier (// or *).*

4.2.5 CellServer to Save Files Remotely

A Java CellServer has been implemented, with which WebTop can communicate to save files remotely on the server. The security restrictions of Java currently does not allow an applet in many popular browsers (Netscape) to read or write to local disks. Therefore, when using a browser which does not allow user defined applet security features, there is no way for the user to have a persistent copy of the cell. The CellServer runs on the same machine from which WebTop is downloaded and therefore the applet can open a socket connection with the CellServer, without violating the security restrictions. In the newer versions of browsers and with digitally signed applets in Java1.1 release, it will be possible to save cells locally, and therefore we will not need the CellServer to obtain a persistent copy of a cell.

The CellServer allows users to store cells remotely at the server's site using login and password verification. Users have the ability to access, modify and delete cells in their respective directories. All cells saved at the server's site are automatically linked as URLs and accessible for use for all other users. Only the owner of the cells has the permissions to modify or delete the cells. Figure 4-6 shows a screen dump of the "Save in URL" dialog box.

The CellServer talks with the client with a simple protocol, in which the client specifies the user, password, cell name and the intended action (Save, Delete or Filter), and the data if necessary. The CellServer uses a flat directory structure for each user and uses a index file `''.index''` to keep track of the files or cells in a particular directory. The CellServer is implemented with a time out mechanism in case of big files or network problems. Finally, the CellServer returns a *status* of the job to the client. WebTop users can make call to the CellServer to save cells as well as save extracted netlists.

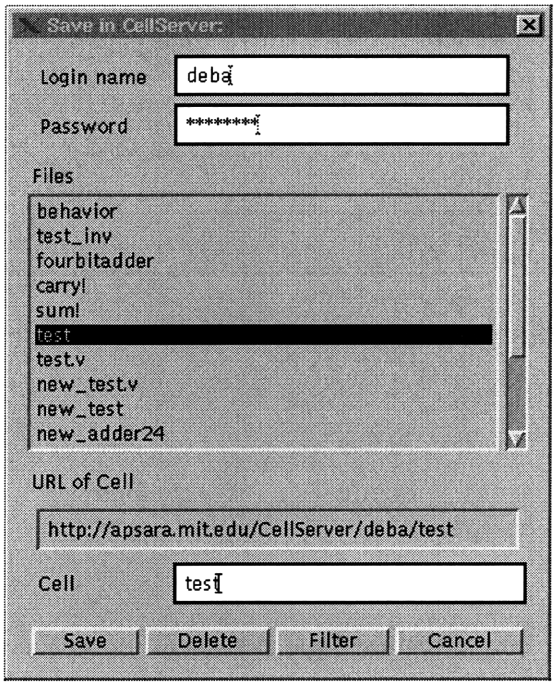

Figure 4-6: The save in URL Dialog

4.2.6 Dynamic Cell Properties

Property name and value pairs can be dynamically added and deleted in a cell. Double clicking on a component pulls up a property window for the sub cell. The top level cell also has a property pull down menu. The property serves as black box properties for PowerPlay and also for future expansion. Any cell instantiated from a parent inherits the properties of the parent. Figure 4-7 shows a screen dump of the property box.

4.2.7 Other Modifications

This section briefly lists some of the modifications to the implementation and functionality of WebTop.

- The Cell format has changed to some extent to make the parsing more effective and also to incorporate the new vectored features. There is also a provision in specifying a version in the cells, such that future versions of WebTop can read and load older versions of cells.

- Cells can also be stored as a persistent *object serialized* form. This takes advantage of the RMI and object serialization mechanisms of Java. Although, saving cells in this raw format makes

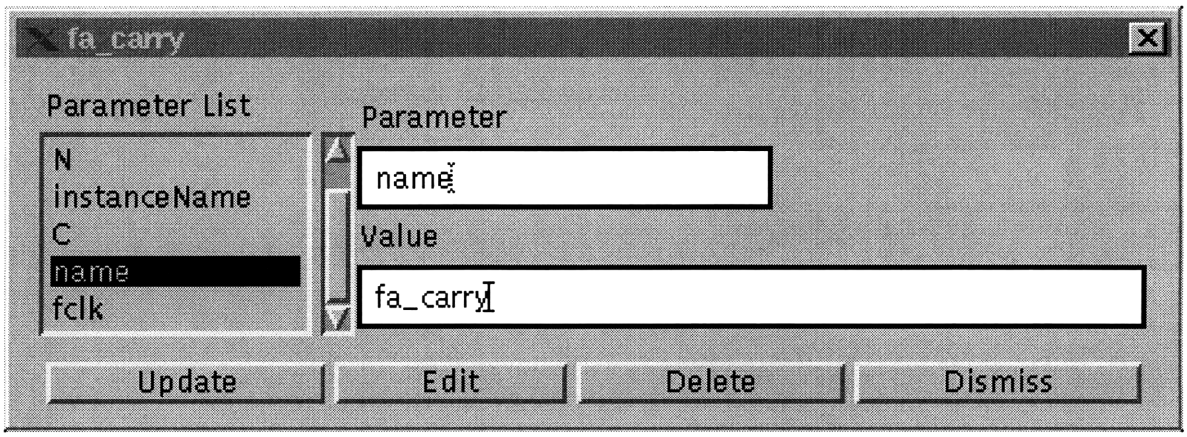

Figure 4-7: Property Box

loading of a cell trivial, we discourage the use of this format because, a textual representation helps in maintaining compatibility. Moreover, the raw object dump tends to make the size of the cells larger, which means a larger bandwidth is required to load cells from web servers.

- Many GUI features have been changed and functionalities enhanced to provide a better look and feel.

- Major changes have been made to the extraction methods, which assigns lexical names to pins (connecting names). The new extraction routine is much more modular and uses the following algorithm:

```
Step 1: Clear all the junction node names.
Step 2: Assign the nodes directly connected to a pin the pin's name
        and all nodes connected to GND as ''0'' (for SPICE).
Step 3: Perform a depth first search on the wires to find
        connected pins.
Step 4: Check if a set of connected wires have a consistent
        name. If no name is provided assign a new name.
Step 5: Assign all the remaining nodes which have not been assigned
        a name so far, and which have at least some cell connected
        to them, different node names.
Step 6: Assign lexical names to all the pins of each sub cell.
```

```
The width of the pins and the connecting nodes are checked.
For vectored instances, use special notations which will
be used by the netlister (*Short:name, *FullBus:name, *Bus:name)
to identify the type of bus connected to a pin of a vector
instance.
```

- Some new functionalities, like extracting to a file, saving extracted netlist in CellServer, have been added.

4.3 Distributed Tool Integration

A major component of the Distributed Microsystem Design Center is its open and transparent integration with different distributed CAD tools. In this section, we will describe WebTop's interfaces to Pythia [20], WebSpice, PowerPlay [19] and a remote Buffer Generator.

4.3.1 Integration with Pythia

Pythia is a CGI based tool and WebTop interfaces with the tool using the mechanism described in Section 3.5. The BIS of Pythia has been presented in Section 3.8. The user extracts a design as a Verilog netlist and provides the simulation instructions. WebTop then creates a multipart/form-data with the netlist as the design file and uses some default parameters for other input parameters of the BIS. WebTop specifies `OutputType` as `URLOutputType`.

The netlist is then submitted to Pythia by making a CGI call to the Web server where Pythia resides. The CGI script executes Pythia, computes the power dissipation and produces a dynamic URL, containing the results as a HTML file. The CGI script then returns the dynamic URL to WebTop. WebTop uses the browser's context to display the dynamic HTML page. The transparent and easy integration of Pythia demonstrates the power of the framework, in applets being able to interface with any CGI based tool.

4.3.2 Integration with WebSpice

We have developed a CGI based point tool **WebSpice** at `http:// apsara.mit.edu/ cgi-bin/ spice/ spice.pl`, which provides Web access to the circuit simulator HSPICE. WebSpice is an independent web tool and can be accessed through the HTML form at: `http:// apsara.mit.edu/ spice.html`. The BIS of WebSpice is very simple and is presented below:

```
// BIS of Pythia
// Location: http://apsara.mit.edu/cgi-bin/pythia/new-pythia.pl

Input:
    <File spice_netlist>
    <Parameter OutputType{HtmlType/BasicOutputType/URLOutputType}>
Output:
    <File spiceOutput>
```

Using WebTop, the user can extract a design schematic into a SPICE netlist. WebTop, then submits the netlist to WebSpice as multipart/form-data conforming to the BIS of WebSpice. WebSpice simulates the spice netlist and sends back the results of simulation to the applet as URL. The applet then displays the resulting URL in the browser.

4.3.3 Integration with PowerPlay

PowerPlay [19] is a Web based CGI tool which helps in system level exploration of power consumption. PowerPlay has been integrated from WebTop with a more complex mechanism. There were two issues which made this integration non-standard. PowerPlay, which is a CGI tool developed at Univ. of California, Berkeley, does not follow the conventions of our framework. Moreover, PowerPlay was located on a server different from that of the codebase of WebTop. Therefore, we had to use the proxy mechanism and also some non-standard interfaces to use PowerPlay. The architecture used is shown in Figure 4-8.

1. The user loads WebTop from the MIT Web server.

2. The user extracts a design using the PowerPlay netlister. WebTop calls a proxy CGI tool at MIT's Web server.

3. The proxy tool takes the netlist and creates a URL of the netlist for PowerPlay to access the design as a URL. The tool then returns the URL to WebTop, which contains the link to PowerPlay, with proper links to the design URL.

4. WebTop requests the browser to load the page of the returned URL from the proxy CGI tool.

5. The browser loads the page from UC Berkeley's Web server, which is a HTML spread sheet of the design, generated by a CGI script of PowerPlay.

6. The user then uses the form generated by PowerPlay to estimate the power, by submitting the spreadsheet to PowerPlay again.

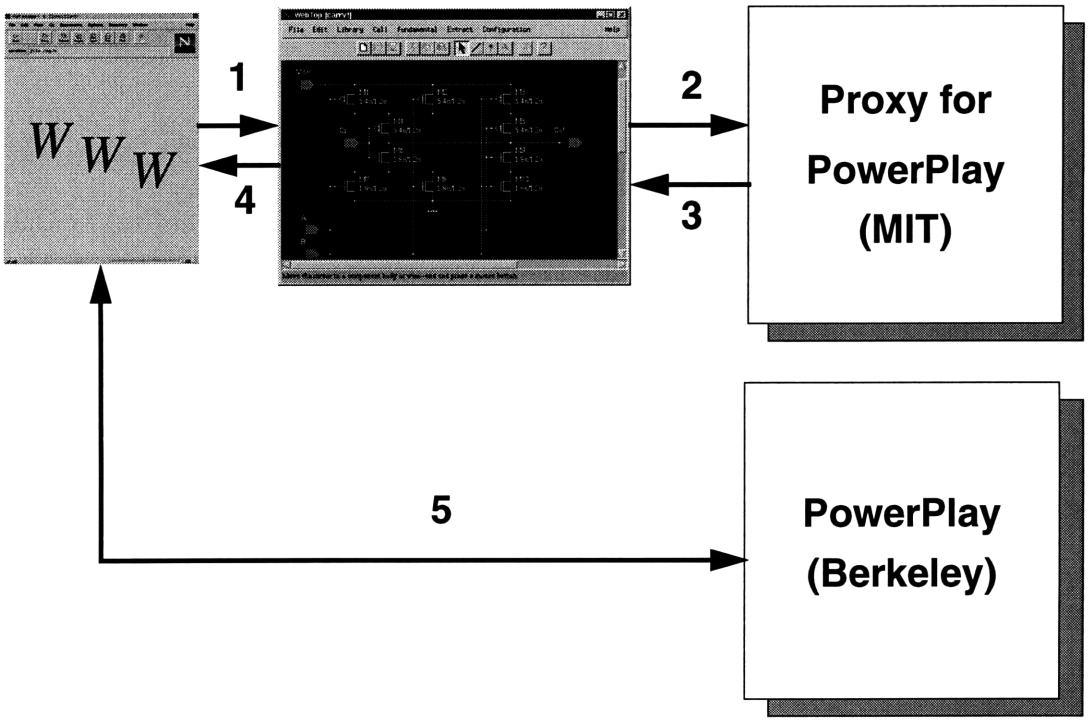

Figure 4-8: WebTop - PowerPlay Integration

The last two steps could be combined, but keeping them separate required minimal modification of PowerPlay and hence was adopted in the prototype architecture.

The PowerPlay netlister in WebTop is still in the experimental stage. The netlist is some form of representation of the system level architecture of the design. The integration also relies on another fact, that the cells used to netlist for PowerPlay must have equivalent models in the PowerPlay database. Currently, we created some equivalent cells for both PowerPlay and WebTop and demonstrated the feasibility of global tool access from WebTop. But what we would ideally like to have is an automated tool which creates equivalent cells both at WebTop's and PowerPlay's database. Issues of representing hierarchy in a meaningful way are in this context are being worked out.

Currently, the integration of any tool from WebTop is done statically. This means to integrate any new CGI based tool, we have to add some code and recompile WebTop. Although, a collections of methods in `WrapUtil.java`, is provided with the source to make the process very simple, it will a good feature to have dynamic invocation of tools. In this case, the applet should read the BIS of the tool and automatically create an input dialog. Once the user enters the inputs, the applet should then wrap the data and invoke the tool. The BIS could contain default values, which the applet might use as inputs.

4.3.4 Integration of a Buffer Generator

In this section, we describe how WebTop interfaces with a distributed object interface, which generates a buffer cell, for a switching power supply. We use Java1.02 RMI mechanisms to invoke the remote object. Although, the example is a simple one, it demonstrates the feasibility of an Object-Web architecture.

The buffer generator is a Java object which generates the schematic of a n stage buffer, where n is the number of stages provided by the user. The interface of the remote generator is listed in Figure 4-9.

```
// Interface RemoteGen
// File: RemoteGen.java

import java.rmi.*;

public interface RemoteGen extends Remote{

      SchematicCell generateBufferCell(String name, int n)
                          throws RemoteException;
}
```

Figure 4-9: Source of RemoteGen.java

A Remote Server then exports the reference of the particular implementation object of the buffer generator with the *Naming* service provided by the RMI mechanism. The source code of the server is listed in Figure 4-10. The object `RemoteGenImpl` implements the interface `RemoteGen`. The clients get a reference to this object by querying the naming service.

We briefly describe the steps for implementing the remote object service using Java RMI.

```
// File: RemoteGenServer.java

import java.rmi.*;
import java.rmi.server.*;
import sun.applet.*;

public class RemoteGenServer{

    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());

        try {
            RemoteGenImpl r1=new RemoteGenImpl();
            Naming.rebind("RemoteGen",r1);
        }
        catch(Exception e){
            System.out.println("Error: " + e);
        }
    }
}
```

Figure 4-10: Source of RemoteGenServer.java

1. We define and write the `RemoteGen` interface, the source of which is provided in Figure 4-9.

2. We write the code for the object which implements the above interface. In this case, it is `RemoteGenImpl.java`, part of which is listed in Figure 4-11.

3. We now compile `RemoteGenImpl.java` to generate the class files.

4. The next step is to generate the `stub` and `skeleton` files for the `UnicastRemoteObject`. We use JDK's `rmic` to generate the files. In UNIX, "`rmic RemoteGenImpl`" generates the stub and skeleton files, `RemoteGenImpl_Stub.class` and `RemoteGenImpl_Skel.class`.

5. Once we have the stubs and skeletons, we write `RemoteGenServer.java`, compile it and execute it to register the remote object with the Naming service. The object runs on the machine `apsara.mit.edu`.

6. The above steps makes the object available for remote clients to invoke methods on them. On the client side, we get a reference to the remote object by incorporating the code listed in Figure 4-12.

```
// File: RemoteGenImpl.java

import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class RemoteGenImpl extends UnicastRemoteObject
    implements RemoteGen{

    public RemoteGenImpl() throws RemoteException{
        super();
    }
    public SchematicCell generateBufferCell(String name, int n)
            throws RemoteException{

         SchematicCell sc;
         /* The code for generating the buffer SchematicCell sc comes here */
         return sc;
    }
}
```

Figure 4-11: Source of RemoteGenImpl.java

The invocation of `generateBufferCell(...)` on the reference to the remote object returns a `SchematicCell`, which contains the schematics of the buffer cell. Figure 4-13 shows the snap shot of a three stage buffer generated by the remote object.

```
// The following code gets a reference to the remote
// object RemoteGen, and invokes generateBufferCell on it.

SchematicCell sc=null;
int n;

try{
    RemoteGen rem=(RemoteGen)
        Naming.lookup("rmi://apsara.mit.edu/RemoteGen");
    sc=rem.generateBufferCell("buffer",n);
}
catch(Exception e) { System.out.println("Error " + e);}
```

Figure 4-12: Client invocation on Remote Object

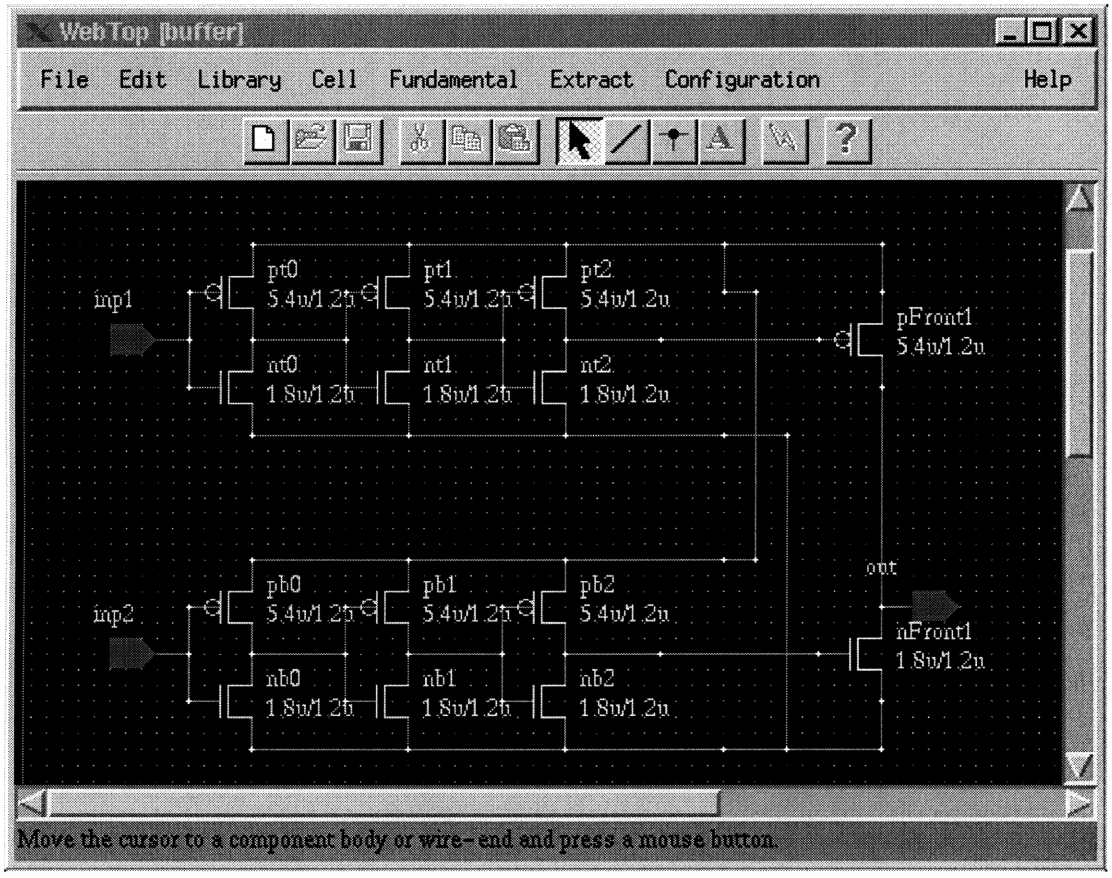

Figure 4-13: Schematics of a Buffer Generated by Remote Object

Chapter 5

Design Examples

Future VLSI design tools must be capable of handling the increased scale of systems-on-a-chip

In this chapter, we present some design examples, using WebTop and the tools in the Distributed Microsystem Design Center. The first example, design and simulation of a 24 bit ripple carry adder is simple enough to illustrate the hierarchical and vectored features of WebTop. The next example, the design of an IDCT chip with more than 150K transistors demonstrates the feasibility of a complex and fairly large VLSI design using WebTop.

5.1 24 bit Ripple Carry Adder

In this section, we present the design steps of a 24-bit adder, and use Pythia to estimate the power dissipation of the adder. Although, this example is simple one, it brings into light the features of WebTop, and the methodology for VLSI design using the Web based tools.

The 24-bit adder uses the block editing capabilities of WebTop. The adder consists of an array of 24 `fulladders`. Each fulladder consists of circuit to compute the sum and carry outputs. In our example, it consists of a `fa_sum`, `fa_carry` and two `inv` cells, as shown if Figure 5-2. Each of the sum, carry and inverter cells have inlined Verilog views. The Verilog modules of the cells are listed in Appendix A.

We first construct the lowest level cells (sum, carry and inverter). Figure 5-1 shows the schematic of the sum cell. The cell has four input pins and one output pin. The verilog view of the cell is also provided as an inlined text in the cell. We note that the order of the pins is important, since

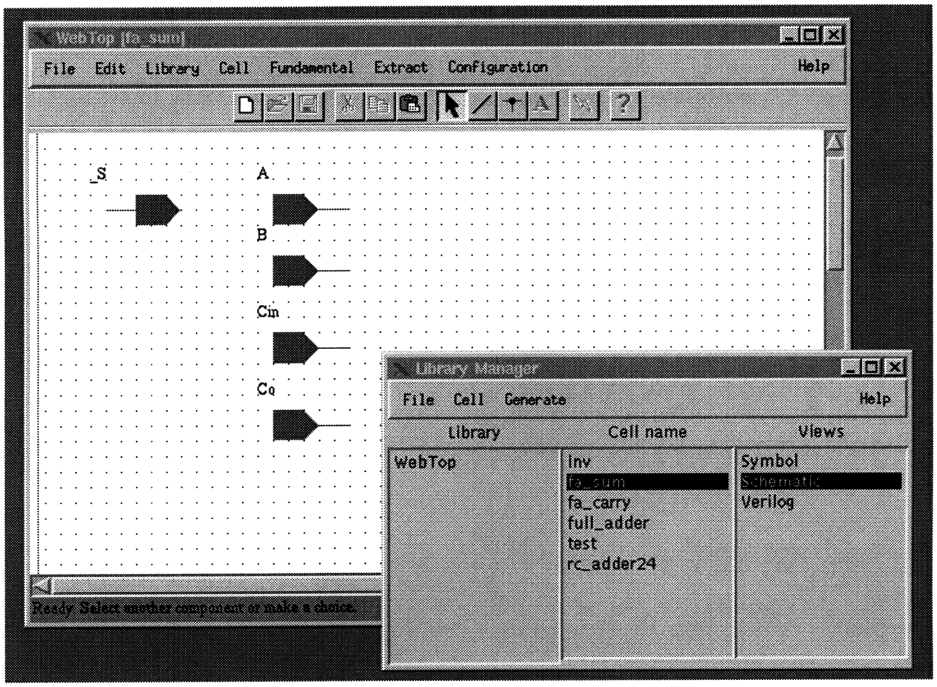

Figure 5-1: Schematic of the Sum Cell

this should match the order in the verilog view of the cell. Similar to the sum cell, we create the carry and inverter also. Once we have the basic cells (blocks), we go ahead and build the fulladder cell. The schematic of the fulladder is shown in Figure 5-2. Once we have the fulladder, we can use the vector instantiation features of WebTop to create a 24 bit adder, `rc_adder24`, the schematics shown in Figure 5-3.

Given all these cells we can construct a top-level `test` cell, shown in Figure 5-4. The user then extracts the `test` cell to obtain the Verilog netlist of the test cell. We then edit the netlist and add the following verilog code to provide the inputs to Pythia.

```
integer i, seeda, seedb;

initial
  begin
    Cin=0;
    seeda= 987;
    seedb=992;
  end

initial
```

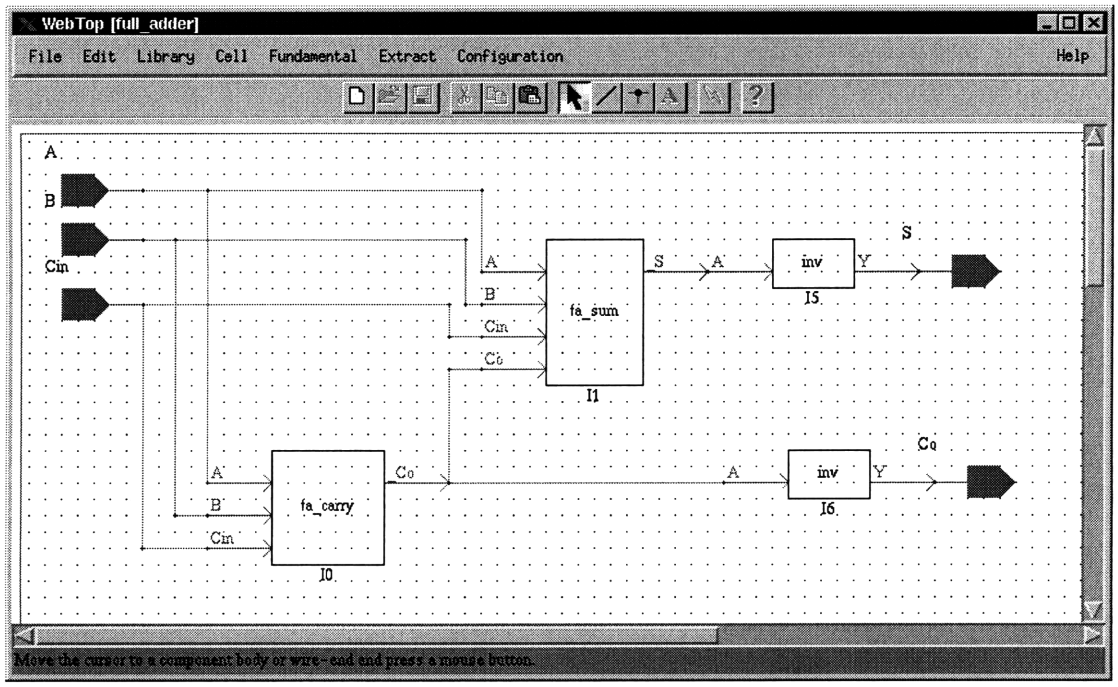

Figure 5-2: Schematic of the Adder Cell

```
  begin
    $setup_pythia(top);
    $run_pythia;
  end

initial
  begin
    for (i=0; i<1000; i=i+1)
      begin
        A= $dist_uniform(seeda, 0, 16777215);
        B= $dist_uniform(seedb, 0, 16777215);
        #100;
        sum= A+B;
        if (sum !== {Cout, S})
          begin
            $display("Error: S= %d, must be %d (A=%d, B=%d)", S, sum, A, B);
           $stop;
          end
      end
  end
```

Finally, we would like to estimate the power dissipation of our 24 bit adder. We pull down Pythia in the *Tools* selection box of the netlister window and submit the netlist for Power Estimation with

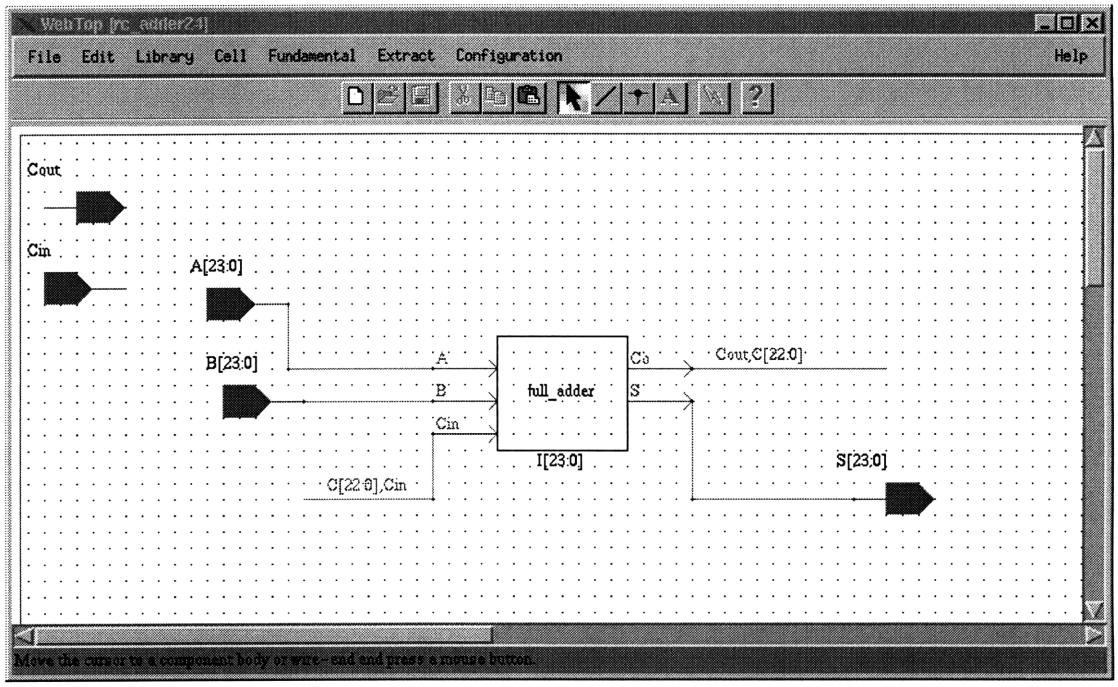

Figure 5-3: Schematic of the 24 bit Adder Cell

Pythia. The applet then get backs the Pythia results and prompts the browser to display the results.

5.2 Design of the IDCT

In this example, we present an IDCT chip [28] for multimedia video application. The schematics of the chip has been ported to WebTop. The chip has been designed, fabricated for ultra low power Inverse Discrete Cosine Transform (IDCT) computation in a typical 3.3v process. The verilog netlist of the chip has been successfully extracted with WebTop's verilog netlister. Table 5.1 summarizes the chip specifications.

The chip involves very complex VLSI design schematics and consists of 160K transistors. The schematic editor is capable of handling such large scale VLSI design. Particularly, the hierarchical features, vectored instantiation and vectored pins of WebTop provide the strength and flexibility to do complex and real life chip designs, like the IDCT chip. The design methodology used to port the IDCT chip demonstrates the feasibility of a distributed VLSI design environment, which is accessible from a Web browser. The user opens up the hierarchical schematic editor, which is

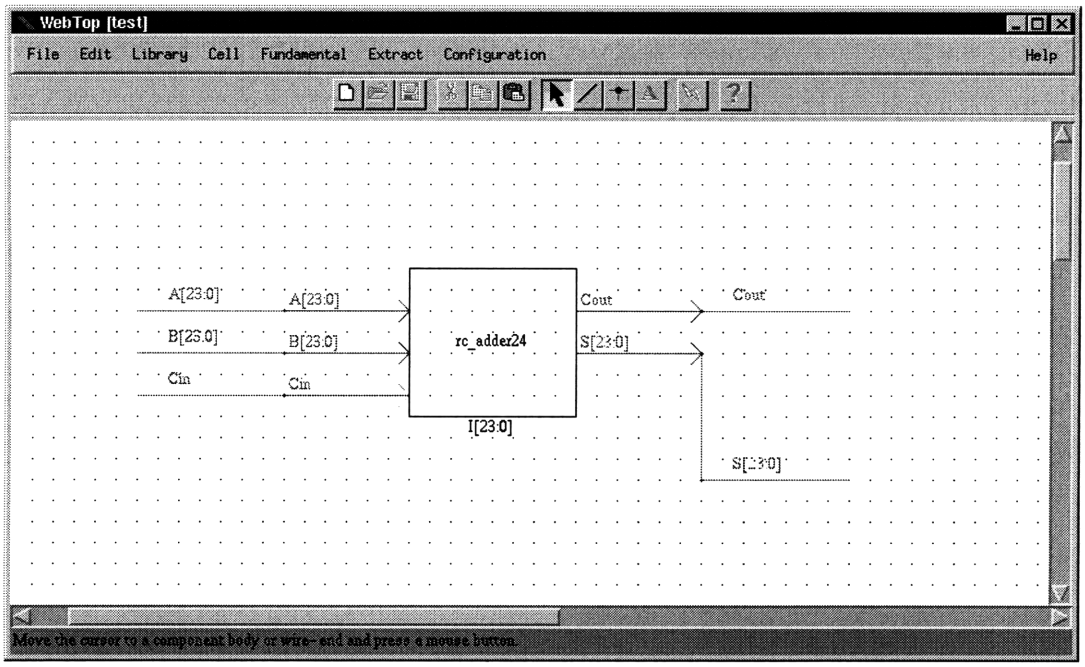

Figure 5-4: Schematic of the Top Level Test cell

Process	$0.7\mu m$ CMOS, 3ML
Supply	1.1-1.9 V
Frequency	5-43 MHz
Power	4.86 mWatts @ 1.32 V, 14 MHz
Area	20.7 mm^2
Transistors	160K

Table 5.1: IDCT Chip Specifications

a Java applet from the Web browser. The user loads different cells from different Web servers as URLs, edits the cell and then uses the CellServer to store cells. The user can then extract the design as a netlist and then makes use of the distributed CAD tools integrated with the editor in a transparent fashion.

We now present the steps followed in designing the IDCT chip using WebTop.

1. We create the schematic of the chip. The schematics are quite complicated and uses a lot of the vectored features of WebTop. The cells are available at the URL `http://apsara.mit.edu/CellServer/idctp/`.

2. Once the block level schematics are available, we create verilog views for the leaf cells.

These views are linked as URLs and are available at `http://apsara.mit.edu/ CellServer/ idctp/VERILOG`. For example, the verilog view for the cell `tspc2_nlatch` is provided as `http:// apsara.mit.edu/ CellServer/ idctp/ VERILOG/ tspc2_nlatch.v`. Many cells like `mux2_pass` have inlined views also. The `mux2_pass` has both a transistor level schematic and a verilog view. Since it has a explicit verilog view, the netlister uses the view instead of the schematic.

3. For simplicity, we did not draw all the cells. The cells which were not created, but used by other cells were put together in `http:// apsara.mit.edu/ CellServer/idctp/ VERILOG/ other_modules.v`, represented by the cell `other_modules` in the top level cell `chip`.

4. Given the schematics of the chip and the constituent cells, we next *extract* the chip in a verilog netlist.

5. The verilog netlist of the chip is then saved in `http:// apsara.mit.edu/ CellServer/ idctp/ VERILOG/ chip.v`.

6. The chip is then simulated using Pythia.

The snap shots of some cells are shown below. The `chip`, part of which is shown in Figure 5-5, consists of a `serializer_s1_new`(Figure 5-6), which again has a `serializer_s1_2reg` (Figure 5-7) as one of the constituent cells. The `serializer_s1_2reg` consists of `mux2_pass` (Figure 5-8), `static2_df`, `inv`, `inv2x`, and some other cells.

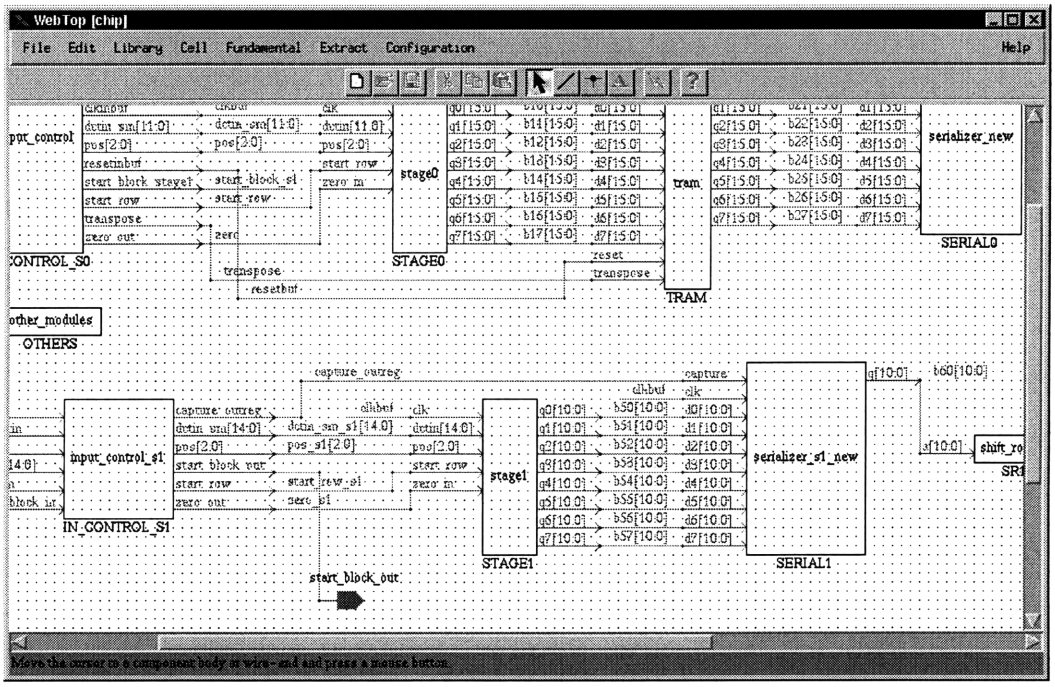

Figure 5-5: Schematic of the Top Level IDCT Chip

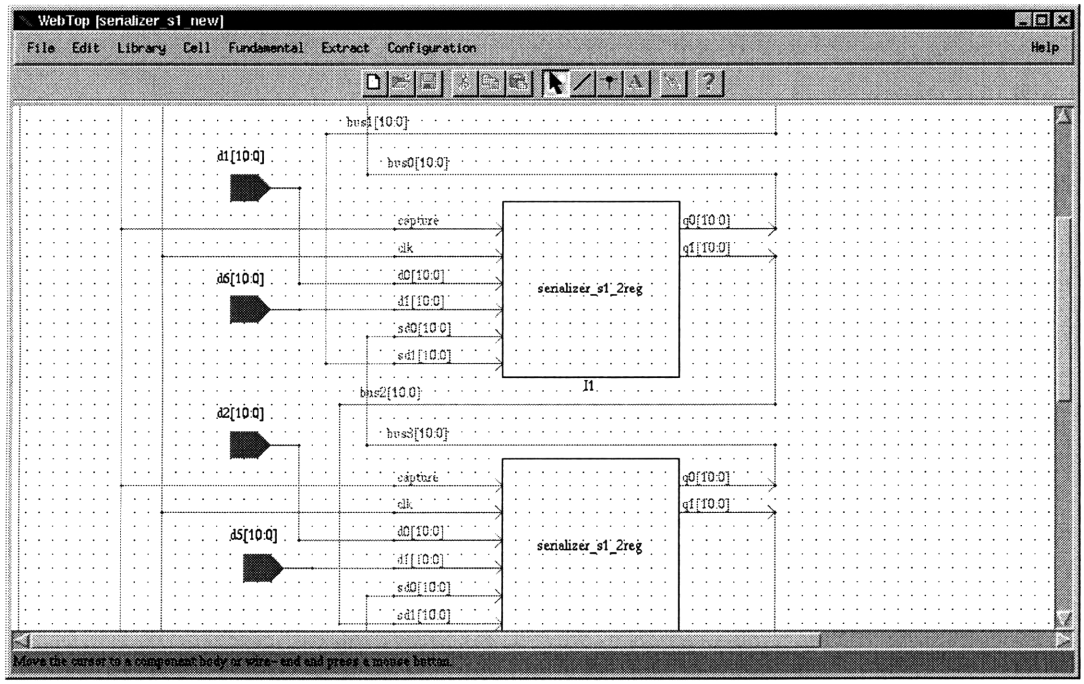

Figure 5-6: Schematic of serializer_s1_new

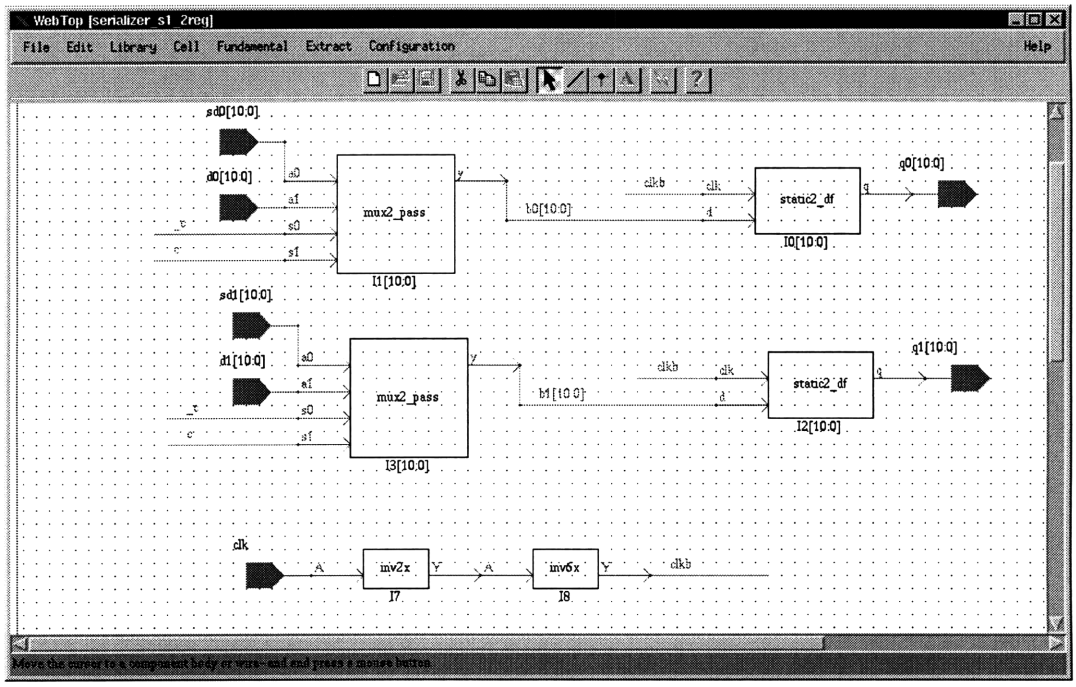

Figure 5-7: Schematic of serializer_s1_2reg

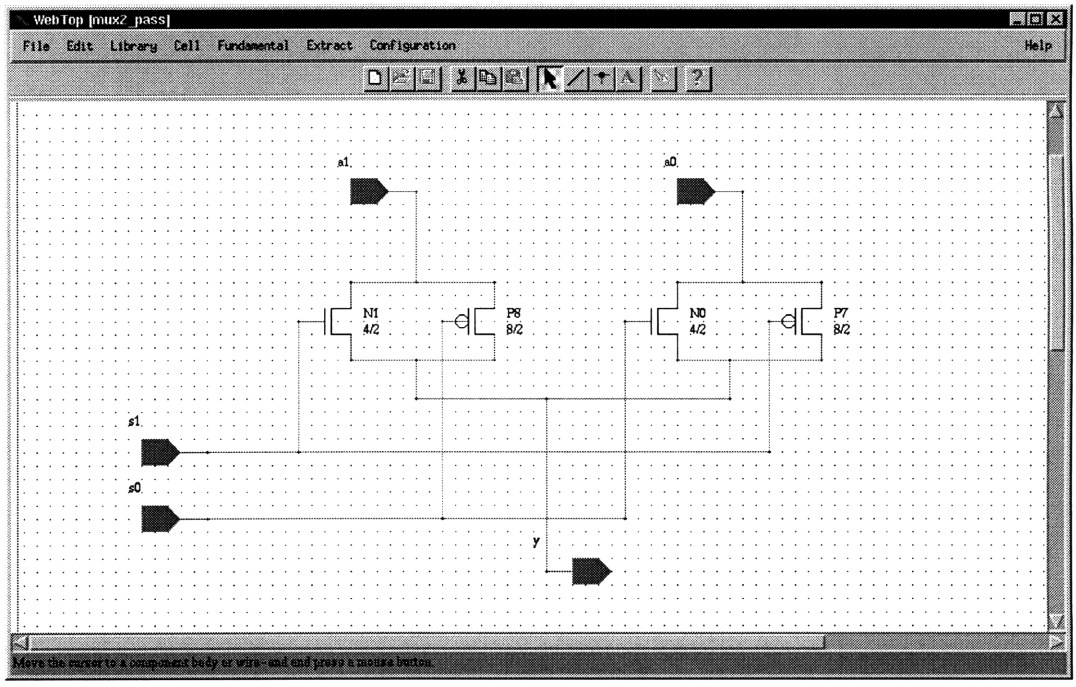

Figure 5-8: Schematic of mux2_pass

Chapter 6

Summary and Future Work

Web-based CAD is the first step towards a globally distributed VLSI design environment

Deep sub-micron technologies enable integration of tens of millions of transistors on a single chip and therefore make it possible to implement an entire system on a chip. Systems-on-a-chips present new challenges to the VLSI CAD community in terms of increased scale and complexity. The high integration levels and the need for diverse expertise at various stages and abstraction of the design process mandates a distributed VLSI design methodology.

The popularity and acceptance of the internet and the World Wide Web makes the Web an attractive platform for distributed applications. The Web as a platform independent way to deliver information has gained strength with the emergence of Java applets being embedded in the Web browsers. In the recent years we have also seen the emergence and maturity of the distributed object technologies. CORBA and Java RMI enables the vision of a transparent and globally distributed object space, which makes it possible to invoke a method on a remote object as if it were a local method invocation. Integrating the object and Web technologies allows us to build an open and interoperable distributed computing environment, which gains strength from the diverse resources available on the internet.

The Object-Web architecture for distributed computing presents a new paradigm for Web based VLSI CAD, by enabling access to design tools and cell libraries irrespective of their geographical or physical availability. In this thesis, we presented a framework for distributed VLSI design which enables hierarchical Web based CAD, by allowing design tools to transparently invoke other tools on Web. Both CGI and the object technologies are important to provide remote access to tools.

Designing the interface of a tool around CGI, CORBA or RMI should take into account how good the fit is with the middleware mechanism. For example, if a tool takes as input an array of bytes or a text file, and returns some parameters as text, CGI mechanisms may be a better fit. But if a tool can capitalize on the strong data typing features of IDL or Java, the object-oriented technologies provides a better abstraction for the tool. For example, *the buffer generator* tool described in Chapter 4, generates a `SchematicCell` object, which can be directly loaded by WebTop and requires no additional reconstruction of the cell from an intermediate data type. To support such a user defined type the object technology (RMI) was a better match. In the case of *Pythia*, which generates a file of data sets, represented as ASCII characters, CGI mechanism was more appropriate. For many VLSI CAD tools, whose input and output interfaces are primarily design files, CGI mechanism provides an easy and inexpensive way to access of the tools remotely.

The major issues concerning Web based CAD include latency, bandwidth, reliability, scalability and security. As regards the network infrastructure, bandwidth requirements tend to be a major bottleneck to ensure fast transfer of huge amounts of data across the network. However, the recent developments in high bandwidth networks (e.g., Gigabit ethernet, optical networks etc.) appear promising in alleviating this bottleneck in the near future. Careful partitioning of tools into Java clients and server tools will help in reducing the latency.

The three tier architecture also includes a cost in the middleware services. But for most of the CAD tools, which are computationally intensive, this extra overhead can be amortized over tool execution time. In some cases, the execution time may run into hours. Such server tools should inform the clients of the initiation of the job and may notify the user the completion of the task by email. These tools, should provide adequate statistics and mechanisms for the clients to query the status and perform additional operations on them, like abortion of the job, if necessary.

The final and possibly the most important concern involving Web based CAD is the security issues involved. As with anything that operates on the internet, data security assumes paramount importance when VLSI design cells represent significant intellectual property resource. A main security issue is the secure transmission of designs and associated data to the server tools. The emerging standards and infrastructure for security on the internet in general, like the public-key infrastructure, such as PGP, SSL (Secure Socket Layer) etc., can be applied directly for the secure encryption of data. Another form of security concern arises from the fact that Web based CAD tools and developers themselves cannot be trusted, and therefore it is desirable to minimize the amount of information that is provided to the tools. The primary challenge for such security, is that CAD users

must divulge information to the CAD tools to perform simulation or optimization on them, yet the users need assurances that this does not let others learn about the details of the design, to protect their intellectual property. Some techniques to address this issue, like netlist flattening, functionality removal, net direction alteration, net removal, circuit partitioning, component simplification and I/O encryption and splitting have been discussed in [7].

Another security and performance issue involves applet trust on the client side. A malicious applet, if trusted, could destroy local data, or get access to classified information on the local disks. Therefore in most browsers, Java applets, which are loaded from the network are not allowed local disk access and network connection with any machine other than the machine. This creates a major bottleneck in the functionality of applets. The Java security issues can also be overcome with digitally signed applets. By attaching a digital signature to the Java code, the origin of the code can be established in a cryptographic-ally secure and unforgeable way. If we specify that we trust an organization, any code that bears the signature of the trusted entity may be run without the usual applet security restrictions.

6.1 Future Work

The Object-Web architecture provides a generic way to integrate tools statically. Both the hierarchical CGI framework and the distributed objects require the caller to know of the specifications (BIS, IDL or Java remote interfaces) at compile time. It will be both interesting and useful to make dynamic invocations possible on tools. Dynamic invocations with the DII (Dynamic Invocation Interface) of the ORB structure of Figure 2-6 could be explored. For the CGI based tools, a methodology for dynamic invocation of CAD tools will be useful. For example, a user of WebTop, should be able to call a tool transparently if he knows the URL of the tool. This should not involve any re-compilation of the applet. It is also possible that all object based tools, implement a global interface `CadToolInterface`. Any client can then get a reference of the remote object and invoke a method of the `CadToolInterface`.

There is a lot of work to be done to make the currently available CAD tools delivered on the Web. Many existing tools could be made available as point tools in the framework using the Web based technologies. We expect a multitude of tools to be available over the Web in the near future. Integrating all the tools in a common framework would provide more value to the VLSI community. This also calls for agreements on standards for data communication and tool protocols between tool vendors and the CAD users.

There is a lot of potential in extending WebTop to support collaborative designs. This is specially important since future systems-on-a-chip designs will necessitate collaborative activities between design groups working on a single chip. WebTop could also be extended by providing a database back end to the central cell storage. Design caching on the client end and maintaining consistency between cached copies and persistent copies of a cell is a challenging aspect of the collaborative framework.

Appendix A

Verilog Listing

A.1 Verilog Listing of Cells in 24 bit Ripple Carry Adder Example

A.1.1 The Sum Cell: fa_sum

```
// Verilog HDL for "fa_sum"

`timescale 1ns/ 100ps

`celldefine

module fa_sum (_S, A, B, Cin, Co);
  output _S;
  input A;
  input B;
  input Cin;
  input Co;

  sum #0.7 I0(_S, A, B, Cin, Co);

    specify
      specparam GateCap$A= 5;
      specparam GateCap$B= 5;
      specparam GateCap$Cin= 5;
      specparam GateCap$Co= 2;
      specparam nDrainCap$_S= 1;
      specparam pDrainCap$_S= 2;
      specparam PythiaID= "FA_SUM";
  endspecify

endmodule

`endcelldefine

// Verilog HDL for "sum"
```

```
primitive sum(s, a, b, cin, co);
  output s;
  input a, b, cin, co;

  table
      // a b cin co   s
      0  0  0  0    : 1;
      0  0  0  1    : 1;
      0  0  1  0    : 1;
      0  0  1  1    : 0;
      0  1  0  0    : 1;
      0  1  0  1    : 0;
      0  1  1  0    : 1;
      0  1  1  1    : 0;
      1  0  0  0    : 1;
      1  0  0  1    : 0;
      1  0  1  0    : 1;
      1  0  1  1    : 0;
      1  1  0  0    : 1;
      1  1  0  1    : 0;
      1  1  1  0    : 0;
      1  1  1  1    : 0;
      x  ?  ?  ?    : x;
      ?  x  ?  ?    : x;
      ?  ?  x  ?    : x;
      ?  ?  ?  x    : x;
  endtable

endprimitive
```

A.1.2 The Carry Cell: fa_carry

```
// Verilog HDL for "fa_carry"

`timescale 1ns/ 100ps

`celldefine

module fa_carry (_Co, A, B, Cin);
  output _Co;
  input A;
  input B;
  input Cin;

  carry #0.7 I0(_Co, A, B, Cin);

  specify
      specparam GateCap$A= 12;
```

```
      specparam GateCap$B= 12;
      specparam GateCap$Cin= 6;
      specparam nDrainCap$_Co= 4;
      specparam pDrainCap$_Co= 8;
      specparam PythiaID= "FA_CARRY";
  endspecify


endmodule

`endcelldefine

// Verilog HDL for "carry"

primitive carry(co, a, b, cin);
  output co;
  input a, b, cin;

  table
      // a  b  cin     co
         0  0  0    :  1;
         0  0  1    :  1;
         0  1  0    :  1;
         0  1  1    :  0;
         1  0  0    :  1;
         1  0  1    :  0;
         1  1  0    :  0;
         1  1  1    :  0;
         x  ?  ?    :  x;
         ?  x  ?    :  x;
         ?  ?  x    :  x;
  endtable

endprimitive
```

A.1.3 The Inverter Cell: inv

```
// Verilog HDL for "inv"

`timescale 1ns/ 100ps

`celldefine

module inv (Y, A);
  output Y;
  input A;

  not #0.2 (Y, A);
```

```
  specify
      specparam GateCap$A= 4;
      specparam nDrainCap$Y= 1;
      specparam pDrainCap$Y= 3;
      specparam PythiaID= "INV";
  endspecify


endmodule

`endcelldefine
```

A.1.4 The Top Level Cell: test

```
//*** TopLevel Cell: test ***
`timescale 1ns/ 100ps
module test(  );
  wire  Cout;
  reg  Cin;
  wire [23:0]  S;
  reg [23:0]  B;
  reg [23:0]  A;
  reg [24:0] sum;

  // Sub Cells start here
  rc_adder24 top( Cout,S[23:0],A[23:0],B[23:0],Cin );

  integer i, seeda, seedb;

  initial
    begin
      Cin=0;
      seeda= 987;
      seedb=992;
    end

  initial
    begin
      $setup_pythia(top);
      $run_pythia;
    end

  initial
    begin
      for (i=0; i<1000; i=i+1)
        begin
          A= $dist_uniform(seeda, 0, 16777215);
```

```
          B= $dist_uniform(seedb, 0, 16777215);
          #100;
          sum= A+B;
          if (sum !== {Cout, S})
            begin
              $display("Error: S= %d, must be %d (A=%d, B=%d)", S, sum, A, B);
             $stop;
            end
        end
    end

endmodule
//*** End TopLevel Module ***
```

Appendix B

Vectored Pin and Bus Connections

The following examples show how the Verilog netlister will netlist schematics for different pin and bus connections. Three cells, *test*, *test_out*, and the top-level cell of *my_cell* have been used in the examples:

```
module test( inp );
    input[3:0]  inp;
    . . . . . .
endmodule

module test_out( out );
    output[3:0]  out;
    . . . . . .
endmodule
```

Case I: In this example (Figure B-1), the junction of cell *test* is not connected to any wire. This implies that the pin is shorted to an unnamed wire. The cell *my_cell* is netlisted as shown in Figure B-2.

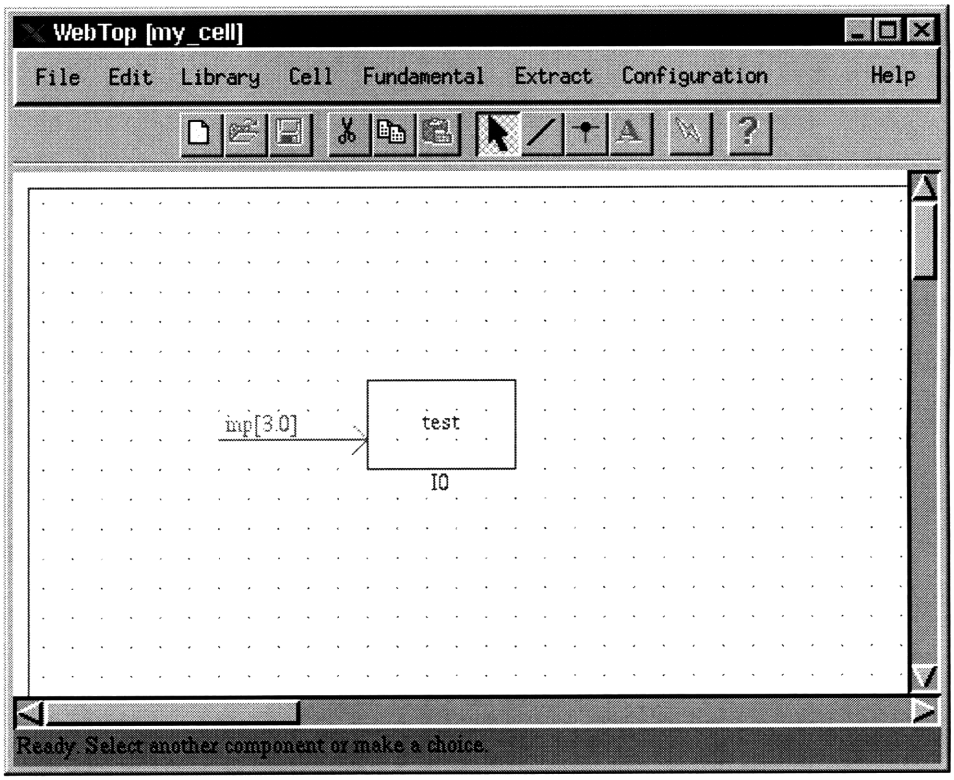

Figure B-1: CaseI: Schematics

```
module my_cell(  );
    // Sub Cells start here
    test IO( _node1,_node1,_node1,_node1 );
endmodule
```

Figure B-2: CaseI: Netlist

Case II: In this example (Figure B-3), the junction of cell *test* is connected to a wire $IN[7:4]$, a wire of similar bit width as the pin. The cell *my_cell* is netlisted as shown in Figure B-4.

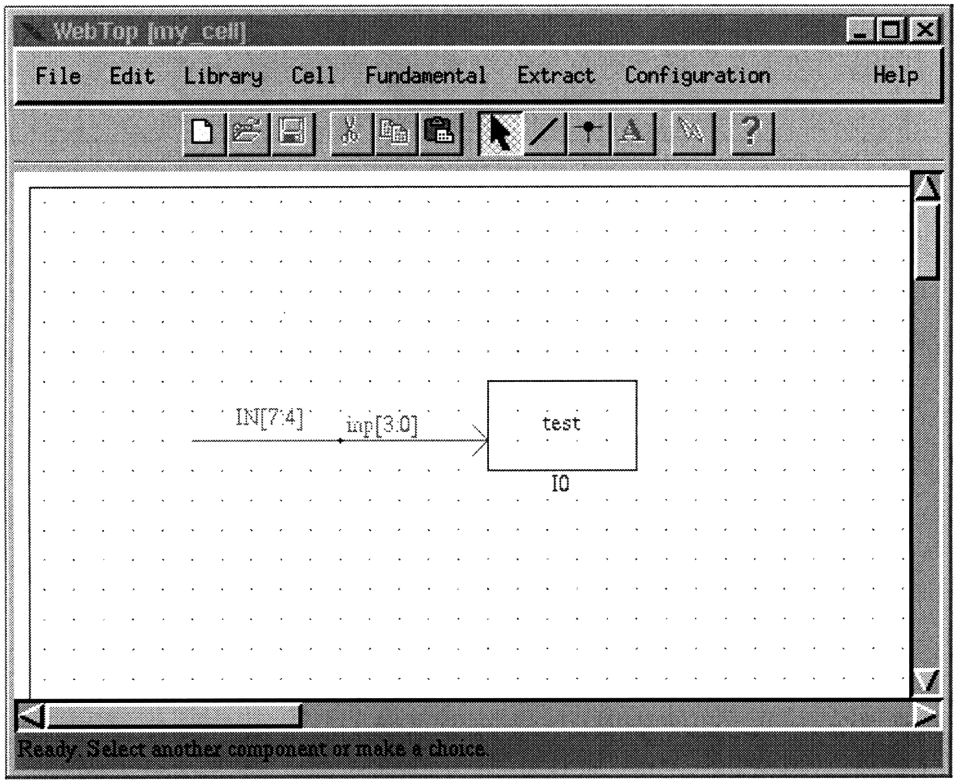

Figure B-3: CaseI: Schematics

```
module my_cell(  );
    // Sub Cells start here
    test IO( IN[7:4] );
endmodule
```

Figure B-4: CaseII: Netlist

Case III: In this example (Figure B-3), the input pin of *test* is connected to the output pin of *test_out*, by a bus *bus*[7 : 4], a wire similar bit width as the connecting pins. The cell *my_cell* is netlisted as shown in Figure B-6.

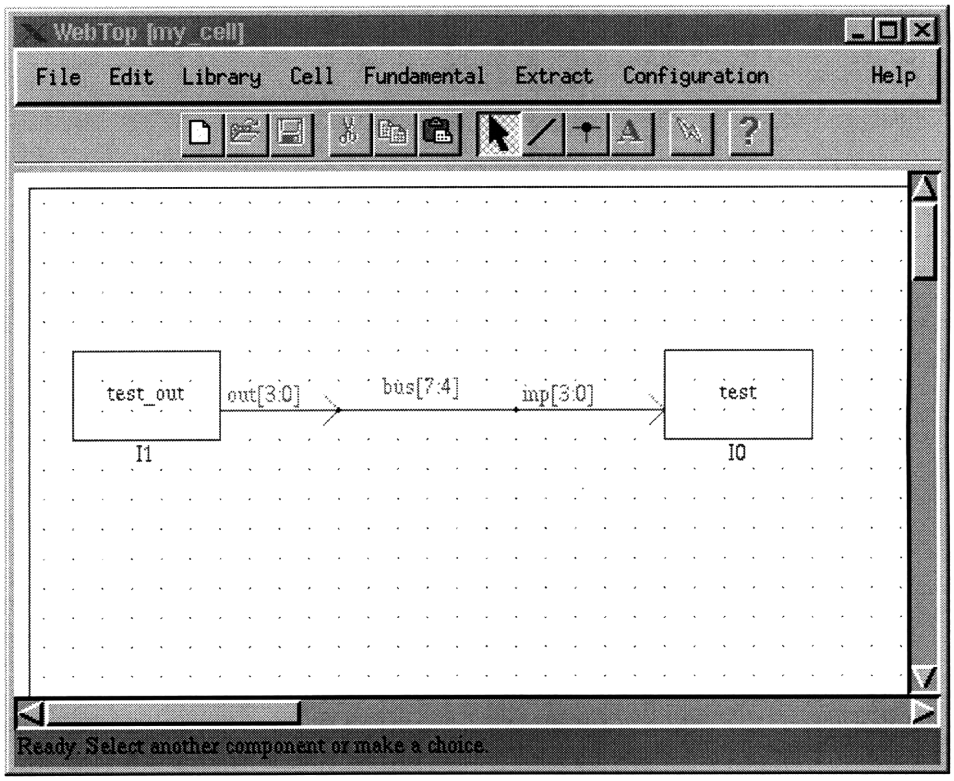

Figure B-5: CaseIII: Schematics

```
module my_cell(  );
    // Sub Cells start here
    test     I0( bus[7:4] );
    test_out I1( bus[7:4] );
endmodule
```

Figure B-6: CaseIII: Netlist

Case IV: In this example (Figure B-7), the input pin of *test* is connected to the output pin of *test_out*, by a junction, which is equivalent to an unnamed wire. The cell *my_cell* is netlisted as shown in Figure B-8.

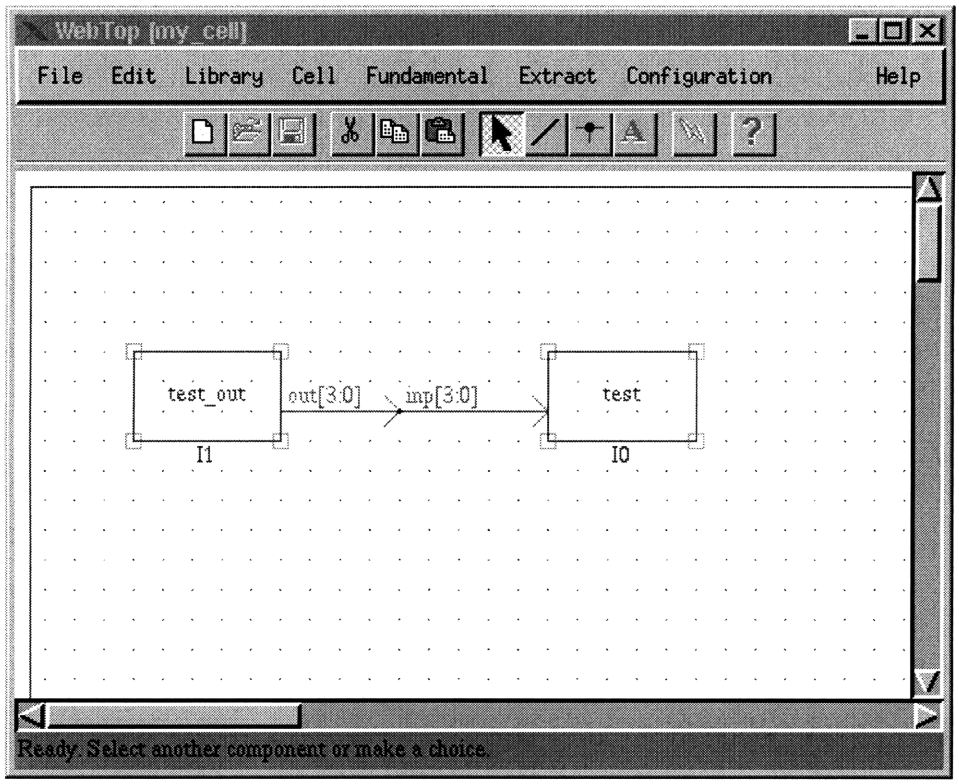

Figure B-7: CaseIV: Schematics

```
module my_cell(  );
    // Sub Cells start here
    test     I0( _node1,_node1,_node1,_node1 );
    test_out I1( _node1,_node1,_node1,_node1 );
endmodule
```

Figure B-8: CaseIV: Netlist

Bibliography

[1] L. Benini, A. Boglio, and G. De. Micheli. Distributed EDA tool integration: the PPP paradigm. *International Conference on Computer design*, 1996. PPP, http://akebono.stanford.edu/users/PPP/.

[2] O Bentz, D Lidsky, and J. M. Rabaey. Information-based Design Environment. *IEEE VLSI Signal Processing VIII*, pages 237–246, Nov 1995.

[3] A. Boglio, L. Benini, G. De Micheli, and B. Ricco. PPP: A Gate-Level Power Estimator - A World Wide Web Application. *Stanford Technical Report No. CSL-TR-96-691*, 1996.

[4] D. R. Chakrabarti, P. G. Joisha, and P. Banerjee. The WADE Project. http://cpdcser.ece.nwu.edu:8080/WADE/wade.html.

[5] Collaborative Benchmarking Laboratory. http://www.cbl.ncsu.edu/.

[6] DigSim: Digital Simulator http://www.lookup.com/Homepages/96457/digsim/index.html.

[7] Scott Hauck and Stephen Knol. Data Security for Web-based CAD. Available at http://www.eecs.nwu.edu/ hauck/webcad.html, Submitted to the DAC, 98.

[8] HTML Specification, http://www.w3.org/MarkUp/.

[9] HTTP Specification, http://www.w3.org/Protocols.

[10] Java 2D Graph Package, http://www.sci.usq.edu.au/staff/leighb/graph/.

[11] N. H. Kapadia, M. S. Lundstrom, Jose' A. B. Fortes, and K. Roy. Network-based simulation laboratories for microelectronics systems design and education. *International Conference on Microelectronic Systems Education, Arlington, Virginia*, page in press, July 1997.

[12] K. Kozminski, B. Duewer, H. Lavana, A. Khetawat, and F. Brglez. REUBEN: A Tcl–Based Reusable Environment Driven by Benchmarking Applications. Technical Report 1995-TR@CBL-03, CBL, ECE Dept., NCSU, Box 7911, Raleigh, NC 27695, October 1995. Also available at http://www.cbl.ncsu.edu/publications/.

[13] H. Lavana and F. Brglez. OmniDesk and OmniFlows: A Platform-Independent Executable and User-Reconfigurable Desktops and Workflows on the Internet. Technical Report 1997-TR@CBL-05-Lavana, CBL, CS Dept., NCSU, Box 7550, Raleigh, NC 27695, Oct 1997. Also available at http://www.cbl.ncsu.edu/publications.

[14] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski. Executable Workflows: A Paradigm for Collaborative Design on the Internet. In *Proceedings of the 34th Design Automation Conference*, pages 553–558, June 1997. Also available at http://www.cbl.ncsu.edu/publications/.

[15] D. Lidsky and J. M. Rabaey. Early Power Exploration - A World Wide Web Application. *Proc. Design Automation Conf, Las Vegas, NV*, pages 27–32, June 1996.

[16] David Liebson. A Schematic Editor and Netlist Extractor in Java. Master's thesis, Massachusetts Institute of Technology, 1997.

[17] Microsystem Design Test Bed, http://web_nt.sainc.com/arpa/msdproject/testbed.htm.

[18] OMG. *The Common Object Request Broker: Architecture and Specifications.* John Wiley and Sons, New York, 1993.

[19] PowerPlay: http://infopad.eecs.berkeley.edu/ lidsky/3POWER/PowerPlay.html.

[20] Pythia, A Verilog based Power Estimation Tool, http:// apsara.mit.edu/ pythia-doc/pythia.html.

[21] RFC 1521, http://203.254.180.2/ ydkim/pub/rfc/rfc1521.txt.

[22] Amin Vahdat, Michael Dahlin, Paul Eastham, Chad Yoshikawa, Thomas Anderson, and David Culler. WebOS: Software Support For Scalable Web Services. http://now.cs.berkeley.edu/WebOS/hotos6.ps, Jan 1997.

[23] Amin Vahdat, Paul Eastham, and Thomas Anderson. WebFS: A Global Cache Coherent Filesystem. http://www.cs.berkeley.edu/ vahdat/www6/www6.html, Jan 1997.

[24] The Vela Project, http://www.cb.ncsu.edu/vela/.

[25] WebOS: http://now.CS.Berkeley.EDU/WebOS/.

[26] WebTop: http://apsara.mit.edu/WebTop/.

[27] The WELD Project, http://www-cad.EECS.Berkeley.EDU/Respep/Research/weld/.

[28] T. Xanthopoulos and A. Chandrakasan. A 1.3V, 5 mW Real Time 2-D IDCT Chip for Video Applications. Internal Group Memo, December 1997.

[29] T. Xanthopoulos, Y. Yaoi, and A. Chandrakasan. Architectural Exploration Using Verilog-Based Power Estimation: A Case Study of the IDCT. *Proc. of the IEEE/ACM Design Automation Conference*, pages 415–420, June 1997.